UMONS
Université de Mons

Faculté Polytechnique

Comparaison des performances d'algorithmes de Machine Learning appliqués à des données Cyber

Travail de Fin d'Études réalisé en centre de recherche (Cylab)

Présenté en vue de l'obtention du grade de Master Ingénieur Civil en Informatique et Gestion

Louis DETRY

POLYTECH MONS

Sous la direction de :
Monsieur Alexandre CROIX (Chercheur au Cylab (ERM))
Professeur Stéphane DUPONT (Promoteur)

Juin 2022

pôle hainuyer

Errata

p. 11 Au lieu de "nombres", il faut lire "nombreux".

p. 13 Au lieu de "$\hat{y}_i$est", il faut lire "$\hat{y}_i$ est".

p. 29 Au lieu de "sont représentés des programmes", il faut lire "sont représentés par des programmes".

p. 38 Au lieu de "époques", il faut lire "epochs" dans le texte et dans le tableau.

p. 55 Au lieu de "nombre de négatif", il faut lire "nombre de négatifs".

p. 75 Au lieu de "Table 3.58 - Résumé des performances obtenues ...", il faut lire "Table 3.58 - Résumé des AUC obtenues ...".

p. 75 Au lieu de "Table 3.59 - Résumé des performances obtenues ...", il faut lire "Table 3.59 - Résumé des AUC-PR obtenues ...".

p. 75 Au lieu de "Table 3.60 - Résumé des performances obtenues ...", il faut lire "Table 3.60 - Résumé des *F-Measure* obtenues ...".

Remerciements

Au Professeur Stéphane Dupont pour son aide et ses conseils tout au long de ce travail.

A Monsieur Alexandre Croix pour sa disponibilité, ses conseils et toutes les explications liées à la Cybersécurité.

A l'Ecole Royale Militaire et plus particulièrement au *Cylab* de m'avoir accueilli et permis de réaliser ce travail dans des conditions optimales.

Résumé

La Cybersécurité, ainsi que l'Intelligence Artificielle, connaissent un réel engouement ces dernières années. En effet, les systèmes de sécurité informatique se dotent de plus en plus d'outils de détection d'intrusion plus efficaces les uns que les autres. L'Intelligence Artificielle, bien connue pour ses capacités d'apprentissage, semble tout à fait capable de se prêter à ce genre de tâche de détection. Le projet MASFAD (*Multi-Agent System for APT Detection*) initié au sein de l'École Royale Militaire a pour objectif de mener des recherches afin de développer un système de détection de potentielles attaques sur un réseau informatique. Plusieurs méthodes de Machine Learning ont déjà été testées comme les algorithmes génétiques ou les réseaux de neurones artificiels afin de faciliter la tâche de l'Homme dans l'analyse des fichiers malveillants. Mais d'autres méthodes comme les arbres de décision ou les séparateurs à vaste marge qui ont déjà fait leurs preuves restaient, jusqu'à présent, inexplorées dans ce projet. C'est l'objet de ce travail réalisé en collaboration avec l'École Royale Militaire.

Après de nombreuses recherches dans la littérature scientifique, qui ne cesse de s'étoffer, plusieurs méthodes de Machine Learning ont été sélectionnées pour faire l'objet de recherches plus avancées comme les séparateurs à vaste marge, les réseaux de neurones, la méthode des K plus proches voisins, les arbres de décision ou encore les méthodes ensemblistes. Chaque méthode testée individuellement, et implémentée en Java afin de faciliter l'intégration au projet MASFAD, a permis de mettre en lumière l'intérêt ou pas de son intégration à un système de détection d'intrusions. Tout comme les recherches qui se multiplient d'année en année, de nombreux jeux de données existent mais présentent, pour la plupart, des limitations liées notamment à leur ancienneté ou au manque de variété dans les données. L'École Royale Militaire a mis au point un jeu de données propre particulier. Ce jeu de données, sur lequel tous les résultats de ce travail sont basés, est constitué de fichiers PHP. Certains d'entre eux contiennent des commandes malveillantes, ils sont alors appelés *WebShell*.

Chaque méthode testée séparément, si ce n'est pour la méthode ensembliste qui permet de combiner plusieurs méthodes afin d'améliorer les résultats, a montré des résultats très prometteurs dans l'ensemble. En effet, les tests réalisés avec les réseaux de neurones permettent d'obtenir des performances très satisfaisantes et améliorent les résultats obtenus auparavant. Les arbres de décision permettent également d'obtenir de bons résultats et présentent un temps d'apprentissage plus avantageux par rapport aux réseaux de neurones. La méthode des arbres de décision est celle dont le temps d'apprentissage est le plus court, de l'ordre de quelques secondes, tandis que l'entraînement des réseaux de neurones se compte en plusieurs dizaines de minutes. La méthode ensembliste ne permet, quant à elle, pas d'améliorer les résultats obtenus avec tous les autres classifieurs.

Table des matières

Chapitre 1

Introduction

Que ce soit pour un usage privé ou professionnel, le matériel informatique occupe une place importante dans notre quotidien. Et, comme tout ce qui est développé par l'Homme, ces outils présentent des failles. De nombreuses personnes mal intentionnées tentent de les exploiter dans le but de nuire et de faire du profit au détriment d'autres individus, institutions, organisations, etc. La Cybersécurité a pour rôle de mettre en place des solutions afin de lutter contre ces attaques et ainsi protéger les différentes parties. La quantité de recherches a explosé dans le domaine ces dernières années et ce travail s'inscrit dans cette dynamique. En effet, les pages qui suivent exposent le travail de recherche réalisé sur la comparaison des performances d'algorithmes de Machine Learning appliqués à des données Cyber. Le travail consiste donc à implémenter différents algorithmes de Machine Learning permettant de détecter diverses attaques informatiques dans le but de comparer leurs performances et d'évaluer les méthodes les plus adaptées pour effectuer cette tâche de détection. Ces algorithmes serviront par la suite dans un projet de plus grande envergure réalisé au sein du *Cyber Defense Lab* (*Cylab*), qui est le laboratoire de cyberdéfense de l'École Royale Militaire (ERM), en partenariat avec lequel a été réalisé ce travail de fin d'études.

A la suite de cette courte introduction, le chapitre 2 fait état de la littérature sur le sujet et présente le jeu de données utilisé tout au long du travail. Le chapitre 3 expose le langage d'implémentation utilisé, les différents algorithmes implémentés, leurs performances ainsi qu'une comparaison de ceux-ci avant de terminer avec une conclusion dans le chapitre 4.

Chapitre 2

État de l'art

2.1 Définitions et concepts

Avant d'entamer la revue de la littérature proprement dite, voici un rappel de quelques notions et concepts relatifs à la cybersécurité.

La cybersécurité est un ensemble de technologies et de procédés qui sont utilisés pour protéger des ordinateurs, des réseaux, des programmes et des données contre les attaques, les accès non autorisés, l'altération ou la destruction [1]. Une de ces technologies est le système de sécurité. Il est constitué d'un système de sécurité réseau et d'un système de sécurité informatique. Chacun de ces systèmes comprend des pare-feux, un antivirus et un système de détection des intrusions (*Intrusion Detection System - IDS*). Il existe une nuance entre ces IDS et les IPS (*Intrusion Prevention System*). Ces derniers ont la particularité de pouvoir bloquer une connexion réseau détectée en fermant certains ports ou en supprimant certains paquets réseaux contrairement à l'IDS qui se charge uniquement de la détection [2].

Les IDS peuvent se baser sur différentes méthodes d'analyse du réseau ([1], [3], [4], [5], [6], [7], [8], [9], [10], [11]) :

- Détection des anomalies basée sur l'analyse de la déviation par rapport à l'utilisation normale d'un système ;
- Détection des abus et des signatures basée sur l'analyse des signatures des attaques en faisant correspondre les données avec des modèles intrusifs connus ;
- Certains articles parlent même d'une troisième catégorie qui est simplement une méthode hybride qui combine les deux modes de détection ([11], [1], [7], [10]).

Ces systèmes possèdent évidemment des avantages et des inconvénients. Les systèmes basés sur la détection des anomalies sont assez largement utilisés car ils sont capables de détecter de nouvelles intrusions (attaques *Zero Day*). Cependant, le nombre de faux positifs (paquets normaux considérés comme des attaques) est plus élevé. Concernant les systèmes basés sur les abus, le taux de fausses alarmes est fortement réduit mais ils ne peuvent pas détecter de nouvelles attaques car les signatures ne sont pas encore connues.

Les attaques sont généralement classées en 4 catégories distinctes ([4], [7], [8], [9], [10] et

[12]) :

- *Probe* ou *Scan* consiste à exploiter un point faible d'un réseau pour accéder à une machine et son contenu ;
- *Denial of Service* (DoS) ou déni de service en français consiste à surcharger un serveur en lui envoyant des requêtes erronées afin de le rendre hors d'usage ;
- *User to Root* (U2R) consiste à exploiter les vulnérabilités d'un système afin obtenir l'accès à la racine d'une machine à partir d'un utilisateur normal sans privilège ;
- *Remote to Local* (R2L) consiste à obtenir un accès local à une machine vulnérable.

2.2 Dataset

Différentes bases de données sont connues et fréquemment utilisées dans le domaine du Machine Learning et de la Cybersécurité. Les paragraphes suivants présentent les principales bases de données rencontrées au fil des recherches.

2.2.1 DAPRA 98/99/00

Le jeu de données DAPRA (*Defense Advanced Research Projects Agency*), sous la direction de DAPRA et AFRL/SNHS (*Air Force Research Laboratory*), a été collecté et publié par le *Cyber Systems and Technology Group* du *Massachusetts Institute of Technology Lincoln Laboratory* afin d'évaluer les systèmes de détection des intrusions réseaux des ordinateurs [1], [7]. Le dataset DAPRA contient un grand nombre de données du trafic et d'attaques. Ce jeu de données peut se diviser en plusieurs sous-jeux de données :

- DAPRA 1998 qui contient 7 semaines d'enregistrement pour l'ensemble d'entraînement et 2 semaines pour l'ensemble de test (les attaques sont divisées selon les 4 catégories présentées dans la partie 2.1 [7]) ;
- DAPRA 1999 qui contient 3 semaines d'enregistrement pour l'ensemble d'entraînement et 2 semaines pour l'ensemble de test (les attaques sont divisées selon les 4 catégories présentées dans la partie 2.1 auxquelles s'ajoute une nouvelle catégorie d'attaques où l'attaquant tente d'ex-filtrer des fichiers spéciaux qui doivent rester sur l'ordinateur de la victime [7]) ;
- DAPRA 2000 qui contient des scénarios d'attaques spécifiques (LLDOS 1.0, LLDOS 2.0.2 et Windows NT).

Ce dernier sous-ensemble est beaucoup moins utilisé que les deux autres.

Le jeu de données DAPRA apparaît sous différentes formes dans de nombreux articles ([4], [5], [7], [13], [3], [1], [10], [2] et [11]).

Les principales caractéristiques des données présentes dans le jeu de données DAPRA sont visibles à la figure 2.1.

Label	Corresponding Features	Description of Features
A_ID	Alert ID	Unique identifier of alert
D_port	Destination Port	Receiver's Port number
Priority	Priority	Describes the Severity of alerts
S_port	Source Port	Sender's port number
Source _IP	Source IP address	IP address of sender
Target_IP	Target IP address	IP address of a receiver
Time	Time	The time when alert is generated

doi:10.1371/journal.pone.0166017.t009

FIGURE 2.1 – Principales caractéristiques des données dans DAPRA [14]

2.2.2 KDD'99

Le jeu de données KDD'99 a été créé pour le *KDD Cup Challenge* de 1999 [4],[7]. Ce dataset se base sur les données présentes dans le jeu de données DAPRA 1998. Les données possèdent des caractéristiques de base dérivées de DAPRA 1998 mais des caractéristiques ont été extraites suite à une analyse plus poussée des données notamment via des fenêtres temporelles. Les presque 5 millions de données présentes dans KDD'99, ensembles d'entraînement et de test confondus, totalisent 41 attributs qui sont repris dans le tableau de la figure 2.2. La différence entre l'ensemble d'entraînement et de test réside simplement dans le nombre de types d'attaques présentes dans chacun d'eux, ce nombre étant respectivement de 22 et 39 ([8], [1]). Ces attaques sont réparties en 5 catégories qui sont les catégories d'attaques citées dans la partie 2.1 à laquelle on ajoute la catégorie des données dites normales [1]. La répartition des attaques étant la suivante :

— 79% pour DoS ;
— 19% pour les données normales ;
— 2% pour U2R, R2L et Probe.

Le jeu de données semble donc déséquilibré [9].

Plusieurs sources ([13], [12],[7],[3], [15]) font état de certains problèmes liés à ce jeu de données. En effet, la redondance des échantillons y est assez importante : 78% pour l'ensemble d'entraînement et 75% pour l'ensemble de test. Un autre problème qui se pose actuellement avec KDD'99 est qu'il ne reflète plus la réalité du réseau et les dernières tendances en termes d'attaques.

C'est le jeu de données le plus largement cité lors des recherches ([4], [5], [12], [6], [7], [8], [9], [13], [3], [1], [10], [16], [2] et [11]) mais étant donné sa taille, de nombreuses recherches effectuent leurs expériences uniquement sur une partie des données choisie, la plupart du temps, aléatoirement.

Category Name	Features Names	Type	Description
Basic Features	P1. Duration	Integer	Duration of the connection (seconds)
	P2. Protocol_type	Nominal	Type of Protocol (TCP, UDP, ICMP etc.)
	P3. Service	Nominal	Network Service (http, telnet, http, others)
	P4. Flag*	Nominal	Connection status (SF, S0, S1, S2, S3, OTH, REJ, RSTO, RSTO,S0, SH, RSTRH, SHR)
	P5. Src_bytes	Integer	Number of bytes sent from source to destination
	P6. Dst_bytes	Integer	Number of bytes received from destination
	P7. Land	Binary	If source and destination IP are identical. It is 1 else 0
	P8. Wrong_fragment	Integer	Sum of Bad checksum packets in a connection
	P9. Urgent	Integer	Some of packets where urgent bit is set 1.
Content Feature	P10. Hot	Integer	Sum of hot actions in a connection (entering a system directory, creating programs and executing programs)
	P11. Num Failed Login	Integer	Number of failed logins in a connection
	P12. Logged in	Binary	1 if successful login else 0.
	P13 Num compromised	Integer	Sum of not found error appearances in a connection
	P14. Root shell	Binary	1 if root shell is obtained else 0
	P15. Su attempted	Binary	Its 1 if su command attempted else 0
	P16. Num root	Integer	Sum of operations performs as a root in a connection
	P17. Num file creation	Integer	Sum of file creations in a connection
	P18. Num shells	Integer	Number of shell prompts
	P19.Num access files	Integer	Sum of operations in control files in a connection
	P20.Num outbound cmds	Integer	Sum of outbound commands in a ftp session
	P21.Is hot login	Binary	If the user is accessing s root , it is set to 1 else 0
	P22.Is guest login	Binary	If the user is accessing s guest, it is set to 1 else 0
Traffic Feature (2s time window)	P23. Count	Integer	Sum of connections to the same destination IP
	P24. Srv count	Integer	Sum of connections to the same destination Port no.
	P25. Serror rate	Real	Percentage of connections that have activated the flag (P4) s0, s1, s2 or s3, among the connections aggregated in count (P23)
	P26. Srv serror rate	Real	Percentage of connections that have activated the flag (P4) s0, s1, s2 or s3, among the connections aggregated in srv_count (P24)
	P27. Rerror rate	Real	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in count (P23)
	P28. Srv rerror rate	Real	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in count (P23)
	P29. Same srv rate	Real	Percentage of connections that were to the same service, among the connections aggregated in count (P23)
	P30. Diff srv rate	Real	Percentage of connections that were to different services, among the connections aggregated in count (P23)
	P31. Srv diff host rate	Real	Percentage of connections that were to different destination machines among the connections aggregated in srv_count (P24)
Traffic Feature (2s time window from dest. to host)	P32. Dst host count	Integer	Sum of connections to the same destination IP address
	P33. Dst host srv count	Integer	Sum of connections to the same destination port number
	P34. Dst host same srv rate	Real	Percentage of connections that were to the same service, among the connections aggregated in dst_host_count (P32)
	P35. Dst host diff srv rate	Real	Percentage of connections that were to different services, among the connections aggregated in dst_host_count (P32)
	P36. Dst host same src port rate	Real	Percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (P33)
	P37. Dst host srv diff host rate	Real	Percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (P33)
	P38. Dst host serror rate	Real	Percentage of connections that have activated the flag (f4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (P32)
	P39. Dst host srv serror rate	Real	Percent of connections that have activated the flag (P4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (P33)
	P40. Dst host rerror rate	Real	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in dst_host_count (P32)
	P41. Dst host srv rerror rate	Real	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in dst_host_srv_count (P33)

FIGURE 2.2 – Caractéristiques des données dans KDD'99 [10]

2.2.3 NSL-KDD

En raison des nombreuses limitations observées par TAVALLAEE et al. [15], abordées dans la section 2.2.2, concernant le jeu de données KDD'99, un nouveau dataset appelé NSL-KDD a été proposé afin de résoudre certains problèmes relatifs à son prédécesseur. Dans ce jeu de données, tous les échantillons ont les caractéristiques de données réseaux. Il contient 22 attaques différentes qui sont classées selon les 4 groupes cités dans la partie 2.1. NSL-KDD couvre les jeux de données KDDTrain+ comme ensemble d'entraînement et KDDTest+ et KDDTest-21 comme ensembles de test. Ces 2 derniers ensembles contiennent 17 attaques supplémentaires par rapport à KDDTrain+. La répartition entre les différents ensembles peut être visualisée dans le tableau de la figure 2.3 [2], [1]. L'ensemble KDDTest-21 est un sous ensemble de KDDTest+ mais est plus difficile à classifier [1]. Cependant, malgré l'intention de résoudre certains problèmes de KDD'99, NSL-KDD possède encore certaines limitations qui sont abordées par MCHUGH [17]

et ne constitue pas une représentation fidèle de la réalité concernant les données réseaux [1].

Ce jeu de données est apparu dans de nombreuses recherches ([12], [7], [13], [3], [1], [10], [16], [2] et [11]) que ce soit simplement pour mener des expériences ou pour constituer un point de référence pour effectuer certaines comparaisons entre différentes méthodes.

Les caractéristiques des données présentes dans le jeu de données sont identiques à celles présentes dans KDD'99 (figure 2.2).

	Total	Normal	Dos	Probe	R2L	U2L
KDD Train^{+}	125973	67343	45927	11656	995	52
KDD Test^{+}	22544	9711	7458	2421	2754	200
KDD Test^{-21}	11850	2152	4342	2402	2754	200

FIGURE 2.3 – Répartition des données dans NSL-KDD [1]

2.2.4 Kyoto 2006+ et New Kyoto 2006+

Le jeu de données Kyoto 2006+ a été construit sur base d'enregistrements réseaux réalisés entre novembre 2006 et août 2009. Ces données ont été obtenues à partir de différents leurres à l'université de Kyoto pour évaluer les IDS. Après cette première version, une autre version, appelée New Kyoto 2006+, est apparue contenant, quant à elle, des données enregistrées entre novembre 2006 et décembre 2015 [13]. Les données de ce dataset contiennent 14 caractéristiques conventionnelles extraites en se basant sur KDD'99 et 10 caractéristiques supplémentaires reprises dans le tableau de la figure 2.4.

Les références à ce jeu de données sont moins fréquentes ([3], [13]), cela s'explique par la relative nouveauté de celui-ci.

Category	No	Field	Description
Basic features	1	**Duration**	The length (number of seconds) of the connection
	2	**Service**	The connection's service type
	3	**Source bytes**	The number of data bytes sent by the source IP address (srcIP)
	4	**Destination bytes**	The number of data bytes sent by the destination IP address (dstIP)
Time based traffic features	5	Count	The number of connections whose srcIP and dstIP are the same to those of the current connection (curConn.) in the past two seconds
	6	Same_srv_rate	% of connections to the same service in Count feature
	7	Serror_rate	% of connections that have "SYN" errors in Count feature
	8	Srv_serror_rate	% of connections that have "SYN" errors in Srv_count$^+$ feature
Host based traffic features	9	Dst_host_count	Among the past 100 connections whose dstIP is the same to that of the curConn. in the past two seconds, the number of connections whose srcIP is the same to that of the curConn.
	10	Dst_host_srv_count	Among the past 100 connections whose dstIP is the same to that of the curConn. in the past two seconds, the number of connections whose service type is the same as that of the curConn.
	11	Dst_host_same_src_port_rate	% of connections whose source port is the same to that of the curConn. in Dst_host_count feature
	12	Dst_host_serror_rate	% of connections that have "SYN" errors in Dst_host_count feature
	13	Dst_host_srv_serror_rate	% of connections that "SYN" errors in Dst_host_srv_count feature
Basic features	14	Flag	The state of the connection at the time the connection was written
Detection features	15	IDS_detection	Whether IDS (Intrusion Detection System) triggered an alert for the connection
	16	Malware_detection	Whether malware, also known as malicious software, was observed in the connection
	17	Ashula_detection	Whether shellcodes and exploit codes were used in the connection by using the dedicated software
	18	Label	Whether the session was attacked or not
Basic features	19	Source_IP_Address	The source IP address used in the session
	20	**Source_Port_Number**	The source port number used in the session
	21	Destination_IP_Address	The source IP address used in the session
	22	**Destination_Port_Number**	The destination port number used in the session
	23	Start_Time	When the session was started
	24	**Protocol**	How long the session was being established

FIGURE 2.4 – Caractéristiques des données dans New Kyoto 2006+ [13]

2.2.5 CICIDS2017

Le jeu de données CICIDS2017 a été publié en 2017 par l'Institut Canadien de Cybersécurité (*Canadian Institute for Cybersecurity - CIC*). Les données d'intrusions labellisées présentes dans CICIDS2017 ont été récoltées sur 5 jours (d'un lundi à un vendredi). Les données récoltées sur le premier jour sont des données dites bénignes donc sans intrusion ; les autres jours, on retrouve des données bénignes mais aussi des attaques. Il totalise donc environ 2,8 millions d'échantillons comprenant chacun 85 caractéristiques [2].

Les références à ce jeu de données [2] sont peu nombreuses étant donné leur récence.

2.2.6 Autres jeux de données

A côté des jeux de données présentés dans les sections précédentes, d'autres ensembles existent comme AFDA ([1]), ISCX2012 ([2], [10]), UNSW-NDIS ([10], [16], [2]) ou encore UNB-CIC(2017) ([16]). Certains de ces jeux de données tentent de régler les problèmes ou limitations qui sont apparus dans d'autres ensembles mais ils restent, malgré tout, moins utilisés que les ensembles plus connus présentés précédemment.

2.2.7 ERM Dataset

Le jeu de données qui sera utilisé dans ce travail contient 23415 fichiers PHP dont 1833 sont des *WebShell* PHP. Les 21582 fichiers restants étant des scripts PHP classiques ne contenant aucune commande malveillante. Une attaque *WebShell* est généralement un script écrit en PHP téléchargé par un pirate sur un serveur qui permet d'exécuter du code malveillant afin d'obtenir l'accès et de prendre le contrôle de ce serveur. En annexe A, vous trouverez un exemple de *WebShell* PHP.

Les différents fichiers sont préalablement traités par des agents remplissant le rôle de détecteur de *WebShell*. Ces différents détecteurs ont été construits lors d'un projet au sein de l'École Royale Militaire. L'objectif du projet, nommé *PHP WebShells Detector*, était de développer des détecteurs de *WebShell* en se basant sur des mécanismes de logique floue, permettant de détecter également les nouvelles menaces. Les recherches, menées au cours de ce projet ont donc aboutit à l'implémentation d'un outil de détection basé sur plusieurs mécanismes différents. En effet, un mécanisme à lui seul ne serait pas assez performant étant donné que chacun se concentre sur un aspect particulier du fichier. Ce système est constitué de 5 mécanismes :

- Signature : qui est basée sur une partie du fichier qui peut-être utilisée pour identifier une éventuelle attaque ou y être assimilée sur base d'un fichier malveillant déjà connu et présent dans une base de données de référence. Si le fichier a été modifié ou est inconnu, il ne serait pas détecté par ce mécanisme ;
- Hachage flou (*Fuzzy hashing*) : se base également sur la comparaison avec d'autres fichiers connus. En effet, ce mécanisme se base sur le fait que 2 fichiers similaires ont des séquences de bits en commun et plus ces séquences de bits seront longues plus les hachages des 2 fichiers seront proches. Un calcul de distance est effectué entre les hachages des 2 fichiers afin de connaître leur similitude ;
- Routines dangereuses : mécanisme qui vérifie si le fichier PHP ne tente pas d'exécuter des routines dangereuses du type *passthru, system , backtick operator*, etc. ou anonymes qui seraient passées comme paramètres de sortie ;
- Obscurcissement : vérifie si le code utilise des caractères non-ASCII, des routines de décodage *base64decode, gzuncompress,* etc. (un fichier malveillant tente de masquer son contenu alors qu'un fichier classique n'en a pas besoin) ;
- Entropie : se base sur le caractère attendu ou non d'un signal. En effet, un signal inattendu sera peut-être significatif tandis qu'un signal attendu sera négligé parmi le grand nombre de signaux similaires. Le principe consiste à calculer la fréquence f_i de chaque caractère et ensuite de calculer l'entropie du fichier de la manière suivante :

$$H = -\sum_{i=0}^{n} f_i * \log_2(f_i) \tag{2.1}$$

Chacun de ces agents donne à chaque fichier un score entre 0 et 1 illustrant la probabilité que ce fichier soit malveillant ou non. Voici un exemple de combinaison des sorties de ces 5 détecteurs : $0.083333333, 0, 0.200310638, 0.147058823, 0.256410256$. Pour cet exemple, la valeur de sortie de chaque détecteur est inférieure à 0,5. Cela indique que tous les détecteurs sont d'accord pour dire que le fichier PHP considéré n'est pas malveillant.

Le jeu de données à disposition peut donc être représenté par un tableau de 5 colonnes, correspondant aux différents résultats obtenus par les agents, et 23415 lignes correspondants au nombre de fichiers PHP.

2.2.8 Dimension des données

La dimensionalité des données est un élément important à prendre en compte dans le domaine du Machine Learning. En effet, la quantité de données est primordiale mais le nombre de caractéristiques qui constituent chacune de ces données l'est également. Dans le domaine de l'apprentissage automatique, cet aspect de la dimension préoccupe de nombres chercheurs si bien que bon nombre d'entre eux évoquent même le fléau de la dimensionalité ([18]). En effet, lorsqu'un modèle doit généraliser avec des données de grandes dimensions, la tâche est souvent plus longue et donne lieu à un modèle plus complexe. En considérant que chaque dimension est séparée en 4 catégories, cela se représente facilement dans l'espace à une dimension (une seule caractéristique dans les données). Si la dimension des données passe à 2, alors le nombre de catégories passe de 4 à 16 et ainsi de suite (figure 2.5). Le nombre de régions de l'espace augmente donc de manière exponentielle par rapport à la dimension des données. Et plus ce nombre de régions augmente, plus la quantité de données devra être importante afin de pouvoir établir une généralisation de la distribution.

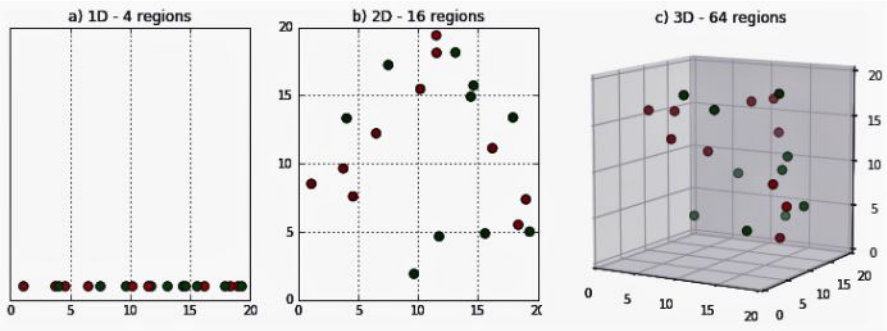

FIGURE 2.5 – Distribution des données en fonction de la dimensionalité [18]

Pour les algorithmes d'apprentissage qui reposent sur le calcul de distance, plus la dimension des données augmente plus ce calcul devient complexe. C'est notamment le cas pour la méthode des K plus proches voisins. La solution à ce problème est alors d'utiliser des méthodes de réduction de la dimensionalité. De nombreuses méthodes existent notamment celle basée sur la corrélation entre les caractéristiques qui est couramment utilisée. Certaines méthodes intègrent directement cette fonctionnalité de réduction de la dimension comme les séparateurs à vaste marge via la fonction de noyau.

Dans le cas des données utilisées dans ce travail, l'espace des caractéristiques a une dimension de 5 ce qui reste raisonnable par rapport à certains jeux de données abordés dans les points précédents dont la dimension peut monter jusqu'à 40. Cependant, il est intéressant de considérer la dimension de l'espace des caractéristiques, peu importe sa valeur.

2.3 Indicateurs de performance

Dans les sections qui vont suivre, différents indicateurs de performance sont utilisés afin d'évaluer les différentes méthodes. Outre leurs valeurs, il est intéressant de connaître les différentes interprétations derrière ces chiffres. La plupart de ces indicateurs se basent sur la matrice de confusion (tableau 2.1).

	Prédit comme positif	Prédit comme négatif
Étiqueté comme positif	Vrai positif (TP)	Faux négatif (FN)
Étiqueté comme négatif	Faux positif (FP)	Vrai négatif (TN)

TABLE 2.1 – Matrice de confusion pour une classification binaire [1]

Pour la classification binaire, la matrice de confusion a une taille 2*2 (tableau 2.1) et pour une classification en n catégories cette matrice a une taille n*n [1]. Voici les différents indicateurs rencontrés ainsi que leurs significations respectives :

— *Accuracy* décrit la proximité ou l'éloignement d'un ensemble donné de mesures par rapport à leur valeur réelle. Cette métrique est intéressante lorsque les différentes classes sont bien équilibrées. En effet, si 97% des instances appartiennent à la classe X et 3% appartiennent à la classe Y et que tous les éléments sont classés dans la classe X, l'accuracy sera de 97% alors que tous les éléments appartenant à Y sont mal classés [7] ;

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

— Précision (*Precision*) correspond à la proportion des éléments correctement classifiés comme appartenant à la classe X sur tous les éléments classifiés comme appartenant à X [7] ;

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

— Rappel (*Recall*) ou taux de vrais positifs (*True Positive Rate - TPR*) correspond à la proportion des éléments correctement classifiés comme appartenant à la classe X sur tous les éléments qui appartiennent à X [7] ;

$$Recall = TPR = \frac{TP}{TP + FN} \tag{2.4}$$

— Taux de faux positifs (*False Positive Rate - FPR*) correspond à la proportion des éléments incorrectement classifiés comme appartenant à la classe X sur tous les éléments qui n'appartiennent pas à X [7] ;

$$FPR = \frac{FP}{FP + TN} \tag{2.5}$$

— Taux de faux négatifs (*False Negative Rate - FNR*) correspond à la proportion des éléments incorrectement classifiés comme n'appartenant pas à la classe X sur tous les éléments qui appartiennent à X. Une valeur élevée signifie que les attaquants réussissent à passer à travers le système de détection d'intrusions [9] ;

$$FNR = \frac{FN}{TP + FN} \tag{2.6}$$

— *F-Measure* traduit l'équilibre entre la précision et le rappel, c'est la moyenne harmonique de la précision et du rappel [8] ;

$$F - Measure = \frac{2 * TP}{2 * TP + FN + FP} = \frac{2 * Precision * Rappel}{Precision + Rappel} \tag{2.7}$$

— AUC (*Area Under Curve*) correspond à l'aire sous la courbe d'une fonction. Il existe deux mesures principales qui se basent sur l'aire sous la courbe :
  — Aire sous la courbe précision-rappel ;
  — Aire sous la courbe ROC (*Receiver Operating Characteristic*) ;
  Dans les deux cas[1], un classifieur idéal a une AUC qui approche de 1, tandis qu'une valeur de 0.5 est comparable à une classification aléatoire [8] ;

— RMSE (*Root Mean Square Error*) présente la différence entre les sorties réelles et les sorties souhaitées sur base de la matrice de confusion [9] ;

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{2.8}$$

— *Kappa-Statistic* est une mesure corrigée du hasard de la concordance entre les classifications et les vraies classes. Une valeur supérieure à 0 signifie que le classifieur fait mieux que le hasard [8] ;

— MAE (*Mean Absolute Error*) mesure l'ampleur moyenne des erreurs dans un ensemble de prévisions, sans tenir compte de leur direction. Elle mesure la précision pour les variables continues [8].

$$MAE = \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|}{n} \tag{2.9}$$

Où $\hat{y}_i$ est la sortie prédite et y_i la sortie réelle.

Lors des différents tests effectués sur les méthodes implémentées dans ce travail (section 3), 3 indicateurs principaux sont utilisés pour exprimer les performances.
Les deux premiers sont l'AUC et l'AUC-PR. Pour ces indicateurs, chaque valeur indiquée lors de la présentation des différents résultats est en réalité une moyenne des valeurs obtenues sur chacun des k blocs de la validation croisée. La validation croisée à k blocs consiste à séparer le jeu de données en k parties. Chaque partie servant chacune à son tour d'ensemble de test et d'entraînement. Donc plus k sera grand, plus l'ensemble d'entraînement sera grand et l'ensemble de test sera petit.
Le troisième indicateur utilisé est la *F-Measure*. Pour cet indicateur, la méthode de calcul utilisée est la micro-moyenne. Elle se base sur la somme des TP, FN et FP obtenus dans les différents blocs plutôt que sur la moyenne des différentes *F-Measure* obtenues dans chacun de ces mêmes blocs, ce qui correspond à la micro-moyenne.

2.4 Méthodes de Machine Learning

Il existe de nombreuses méthodes qui sont attribuées au Machine Learning à l'heure actuelle. Ces méthodes peuvent se baser sur trois axes d'apprentissage importants :

1. Dans ce travail, lorsque l'abréviation AUC est utilisée seule, elle fait référence à l'aire sous la courbe ROC (AUC ROC) tandis que lorsqu'une référence est faite à l'aire sous la courbe précision-rappel, l'abréviation AUC-PR est exclusivement utilisée.

— Apprentissage supervisé ;
— Apprentissage non-supervisé ;
— Apprentissage par renforcement.

L'apprentissage est dit supervisé lorsque, durant la phase d'entraînement, les labels des échantillons sont connus et qu'ils sont utiles afin d'améliorer progressivement la classification. Les méthodes non-supervisées n'utilisent, quant à elles, pas les labels des données et les modèles tentent de trouver eux-mêmes une certaine structure dans les données afin de les classer. L'apprentissage par renforcement consiste, au fur et à mesure de l'apprentissage, à accorder certaines "récompenses" ou "punitions" selon les actions effectuées. Les sections qui suivent présentent les méthodes les plus souvent rencontrées ainsi que les différents résultats inhérents à ces méthodes.

2.4.1 Classifieur bayésien naïf

Le classifieur bayésien naïf se rapporte à la catégorie des classifieurs probabilistes. Il applique le théorème de Bayes (équation 2.10) à la classification. La première étape de cette classification consiste à déterminer le nombre total de classes dans les données et à calculer les probabilités de chaque classe : $\mathbb{P}(C_k)$. La probabilité conditionnelle est ensuite calculée sur chacun des attributs qui constituent les données : $\mathbb{P}(f_i|C_k)$. Les données utilisées peuvent être discrètes ou continues [9]. Le terme naïf est utilisé car ce classifieur s'appuie sur deux hypothèses simplificatrices [4], [7], [10] :
— Les attributs prédictifs sont conditionnellement indépendants ;
— Aucun attribut caché ou latent n'influence le processus de prédiction.

Théorème 1 (Théorème de Bayes) *Soient $B_i \in \mathcal{A}, i \in I$, des évènements totalement exclusifs tels que $\mathbb{P}[B_i] \neq 0, \forall i \in I$. Soit $A \in \mathcal{A}$ tel que $\mathbb{P}[A] \neq 0$. Alors $\forall j \in I$,*

$$\mathbb{P}[B_j|A] = \frac{\mathbb{P}[A|B_j]\mathbb{P}[B_j]}{\mathbb{P}[A]} \tag{2.10}$$

Les probabilités conditionnelles d'appartenance à une classe, étant donné les attributs, qui forment le modèle du classifieur peuvent s'exprimer comme [7] :

$$\mathbb{P}[C|f_1, ..., f_m] \tag{2.11}$$

Le classifieur attribue une étiquette de classe selon :

$$y(f_1, ..., f_m) = \underset{x_{k\in(1,...,K)}}{\operatorname{argmax}} \mathbb{P}[C_k] \prod_{i=1}^{m} \mathbb{P}[f_i|C_k] \tag{2.12}$$

où :
— m est le nombre de caractéristiques ;
— K est le nombre de classes ;
— f_i est la i^{ieme} caractéristique ;
— C_k est la k^{ieme} classe ;

— $\mathbb{P}[C_k]$ est la probabilité à priori de C_k ;
— $\mathbb{P}[f_i|C_k]$ est la probabilité conditionnelle d'une caractéristique f_i étant donné la classe C_k.

L'avantage de ce classifieur est que l'apprentissage peut être réalisé en temps linéaire [7]. Cependant, l'hypothèse d'indépendance des attributs qui, ici, suggère l'indépendance des caractéristiques n'est pas totalement respectée car, pour ne citer que le jeu de données KDD'99, les caractéristiques des données sont fortement dépendantes. Les classifieurs bayésiens naïfs cachés relâchent cette hypothèse [10].

Les classifieurs bayésiens peuvent être représentés comme des réseaux bayésiens (*Bayesian Network - BN*) exposés dans la section 2.4.9 ou des réseaux de croyances (*Belief Network*) [9].

Concernant les performances obtenues en utilisant la méthode des classifieurs bayésiens naïfs, visibles dans le tableau 2.10, l'utilisation du jeu de données KDD'99 avec un nombre d'attributs limités sélectionnés sur base de la corrélation (*Correlation Feature Selection - CFS*) permet d'obtenir de très bons résultats comme une AUC de 0,999 ou un taux de faux positifs de 0%. Cela fait état d'un classifieur très proche d'un classifieur idéal. On peut aussi constater aux lignes 7 et 8 que ce modèle a été utilisé pour détecter les mauvaises utilisations et les anomalies indépendamment l'une de l'autre, toujours sur le même jeu de données, et la méthode semble donner de meilleurs résultats pour la détection des mauvaises utilisations.

	Article	Dataset	Précision	Accuracy	Recall	Autres indicateurs	Commentaires
1	[3]	New Kyoto 2006+	91,67%	96,72%	91,67%		
2	[12]	NSL-KDD avec tous (41) les attributs	/	70,88% [2]	/		
3	[12]	NSL-KDD avec certains (15) attributs sélectionnés avec *Correlation-based Feature Selection* (CFS)	/	74,28%	/		
4	[9]	Partie de KDD'99	98,80%	91,23%	98,80%	AUC = 0,969 ; RMSE = 0,0872 ; FPRate = 0,2% ; FNRate = 8,5%	
5	[8]	10% KDD'99 avec tous les attributs	98,90%	92,75%	92,70%	AUC = 1 ; FPRate = 0 % ; F-Measure = 0,949 ; MAE = 0,0063 ; Kappa-Stat = 0,8802	
6	[8]	KDD'99 avec 11 attributs sélectionné avec CFS	99,00%	96,16%	96,20%	AUC = 0,999 ; FPRate = 0 % ; F-Measure = 0,973 ; MAE = 0,0037 ; Kappa-Stat = 0,9539	
7	[7]	KDD'99 (mauvaise utilisation)	/	93,75% [2]	/	FPRate = 3% ;	
8	[7]	KDD'99 (anomalie)	/	60% [2]	/		
9	[2]	NSL-KDD avec tous (41) les attributs	/	69,80%	30,10%	FPRate = 1,3% ; F-Measure = 0,457	
10	[2]	CICIDS2017	/	62,10%	99,40%	FPRate = 49,3% ; F-Measure = 0,492	
11	[4]	10% KDD'99 (8020)	/	/	64,38% [2]		
12	[4]	10% KDD'99 (8416)	/	/	63,60% [2]		
13	[4]	10% KDD'99 (8812)	/	/	64,28% [2]		
14	[10]	KDD'99 avec tous (41) les attributs (mauvaise utilisation)	/	/	51,37%		

TABLE 2.2 – Performances des classifieurs bayésiens naïfs

2.4.2 Arbre de décision

Un arbre de décision est un classifieur qui permet de classer des données à l'aide d'une séquence de décisions. Ces décisions sont représentées sous la forme d'une structure arborescente (figure 2.6). Chaque noeud racine ou intermédiaire dans l'arbre représente un attribut et

2. La valeur a été calculée en effectuant la moyenne des résultats obtenus sur les différentes catégories d'attaques présentées dans la partie 2.1.

chaque branche descendante de ce noeud représente les valeurs possibles de cet attribut. Les noeuds feuilles représentent, quant à eux, des catégories de classification [4], [5], [1]. Les arbres de décision peuvent être utilisés avec des données discrètes et continues. Il existe plusieurs algorithmes connus qui se basent sur les arbres de décision comme ID3, C4.5, CART, etc. [10], [1]. L'algorithme ID3 utilise une approche de recherche gourmande. Les attributs sont sélectionnés en fonction du critère de gain d'information [10]. La particularité de l'algorithme C4.5, qui est une évolution de ID3, est que, à chaque noeud, il choisit la caractéristique qui sépare le mieux les données. L'attribut choisi est celui qui a le gain d'information le plus élevé [7]. La complexité temporelle des arbres de décision est de $O(mn^2)$ où n représente le nombre de données et m le nombre de caractéristiques [11].

L'implémentation des arbres de décision est relativement simple et permet une haute précision de classification. Cependant, un désavantage important est que, pour les données qui possèdent des attributs où différentes valeurs sont possibles, le gain d'information est biaisé en faveur des attributs qui ont un grand nombre de valeurs possibles. Pour les petits arbres de décision, il est assez facile d'extraire les règles qui permettent la classification mais la tâche se complique lorsque leur taille augmente. Il est toutefois possible d'obtenir de plus petits arbres en élaguant les plus grands et ainsi faciliter cette extraction. Les grands arbres de décision présentent aussi des limitations en termes de généralisation contrairement à leurs homologues de petite taille [7].

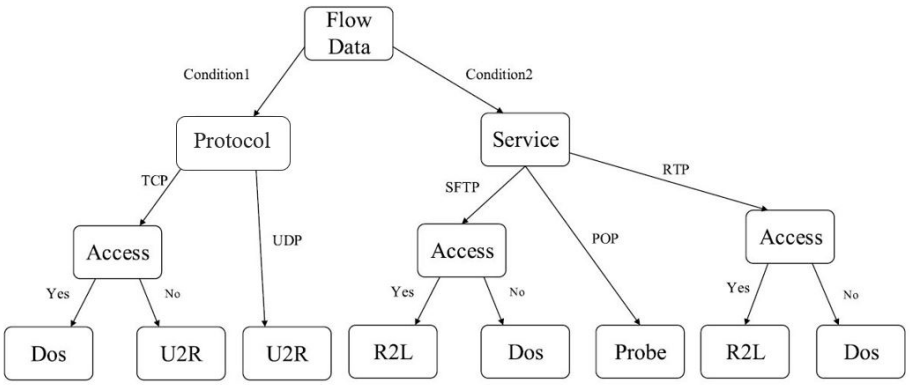

FIGURE 2.6 – Exemple d'arbre de décision pour la classification d'attaques [1]

Lorsqu'on regarde le tableau des performances 2.3 des méthodes utilisant les arbres de décision, les indicateurs sont généralement bons voire très bons. Aux lignes 9 et 10, les résultats obtenus sur le jeu de données KDD'99, en utilisant l'algorithme particulier J48, sont parmi les plus élevés de ce tableau. Contrairement aux classifieurs bayésiens naïfs, sur base des lignes 9 et 10, les résultats semblent légèrement plus élevés en conservant tous les attributs des données bien que cette comparaison soit délicate étant donné qu'un des 2 résultats se base sur l'entièreté du jeu de données tandis que l'autre se base sur une partie de celui-ci.

	Article	Dataset	Précision	Accuracy	Recall	Autres indicateurs	Commentaires
1	[11]	KDD'99 (mauvaise utilisation)	/	99,96%	/		
2	[11]	KDD'99 (hybride)	/	86,29%	78,00%		
3	[11]	NSL-KDD (hybride)	91,15%	90,30%	90,31%		
4	[12]	NSL-KDD avec tous (41) les attributs	/	80,60% [2]	/		J48
5	[12]	NSL-KDD avec certains (15) attributs sélectionnés avec CFS	/	85,24% [2]	/		J48
6	[9]	Partie de KDD'99	98,90%	93,10%	93,10%	AUC = 0,969 ; RMSE = 0,0763 ; FPRate = 0,5% ; FNRate = 6,3%	J48
7	[8]	10% KDD'99 avec tous les attributs	99,80%	99,76%	99,80%	AUC = 1 ; FPRate = 0,1% ; F-measure = 0,997 ; Kappa Statistic = 0,9959 ; MAE = 0,0014	
8	[8]	KDD'99 avec 11 attributs sélectionnés avec CFS	99,70%	99,75%	99,97%	AUC = 1 ; FPRate = 0,1% ; F-Measure = 0,997 ; Kappa Statistic = 0,9957 ; MAE = 0,0016	
9	[8]	10% KDD'99 avec tous les attributs	100,00%	99,96%	100,00%	AUC = 1 ; FPRate = 0% ; F-measure = 1 ; Kappa Statistic = 0,993 ; MAE = 0	J48
10	[8]	KDD'99 avec 11 attributs sélectionnés avec CFS	99,90%	99,94%	99,90%	AUC = 1 ; FPRate = 0% ; F-measure = 0,999 ; Kappa Statistic = 0,999 ; MAE = 0,0001	J48
11	[13]	New Kyoto 2006+ avec partie des attributs sélectionnés selon *Random Forest* (RF)	96,00%	96,00%	96,00%		
12	[10]	KDD'99 avec tous (41) attributs (mauvaise utilisation)	/	/	47,32% [2]		
13	[10]	KDD'99 (mauvaise utilisation)	/	/	97,27% [2]		C4.5
14	[10]	KDD'99 avec certains (23) attributs sélectionnés avec Séparateur à vaste marge (SVM) et recuit simulé (*Simulated Annealing - SA*) (mauvaise utilisation)	/	99,96%	92,26% [2]		
15	[2]	NSL-KDD avec tous (41) les attributs	/	91,90%	86,80%	FPRate = 3,6% ; F-Measure = 0,91	
16	[2]	CICIDS2017	/	97,90%	98,70%	FPRate = 2,2% ; F-Measure = 0,946	
17	[1]	NSL-KDD avec 14 attributs sélectionnés avec CFS	/	90,30%	/		
18	[1]	KDD'99	99,98%	96,65%	/	FPRate = 13,6%	Elagage arbre avec *Swarm Optimization Multi Objectif*
19	[1]	KDD'99	99,89%	91,94%	/	FPRate = 0,81 %	Elagage arbre avec *Swarm Optimization Multi Objectif*
20	[1]	KDD'99 et NSL-KDD	98,45%	/	/	FPRate = 1,55 %	C4.5 avec élagage
21	[1]	KDD'99 avec nombre d'attributs limités sélectionnés avec *Gain Ratio*	/	94,6%	/		
22	[1]	Netflow	/	99%	/		
23	[1]	Netflow	/	84,7%	/		Prend en charge la détection des attaques APT sur base du code malveillant
24	[4]	10% KDD'99 (8020)	/	/	52,18% [2]		
25	[4]	10% KDD'99 (8416)	/	/	51,60% [2]		
26	[4]	10% KDD'99 (8812)	/	/	52,58% [2]		
27	[6]	Partie de KDD'99	/	/	95,00%	FPRate = 1 %	
28	[16]	KDD'99	/	97,64%	89,78% [2]	FPRate = 0,56% [2], F-Measure = 0,8910 [2]	C4.5
29	[16]	NSL-KDD	95,93% [2]	98,80%	/	FPRate = 0,32% [2] , F-Measure = 0,9624 [2]	C4.5

TABLE 2.3 – Performances des arbres de décision

2.4.3 Forêt aléatoire

Les forêts aléatoires sont simplement une combinaison de plusieurs arbres de décision exposés dans la section 2.4.2. En effet, plusieurs arbres sont construits sur bases de différents sous-ensembles des données d'origine. Une fois que la forêt est formée, chaque nouvelle donnée qui doit être classifiée passe dans chacun des arbres qui vote sur la classe de la donnée. Pour choisir la classe finale de l'objet, celle qui a récolté le plus de votes est retenue, cette méthode est appelée méthode du vote majoritaire [4], [9]. La combinaison des résultats via un vote pondéré

en attribuant des poids différents à chaque arbre est également possible [7].

Les avantages de cette méthode sont le nombre de paramètres de contrôle qui est très réduit et sa résistance à un ajustement excessif. Une augmentation du nombre d'arbres permet également une diminution de la variance du modèle. Les inconvénients sont une faible interprétabilité du modèle et une perte de performance due à des variables corrélées [7].

Pour les forêts aléatoires, il semble intéressant, au vu de la comparaison des lignes 1 et 2 du tableau des résultats 2.4, de sélectionner seulement une partie des attributs des données. En effet, les résultats obtenus lorsque seuls 15 attributs sont retenus sur 41 de l'ensemble des données NSL-KDD sont meilleurs que ceux obtenus en conservant l'entièreté d'entre eux. La ligne 4 montre également de bons résultats lorsqu'une partie des attributs est conservée sur le jeu de données New Kyoto 2006+, bien que le tableau ne permette pas de point de comparaison avec l'ensemble des attributs. Concernant les méthodes de détection utilisant les forêts aléatoires, elles semblent plus performantes lorsqu'elles sont intégrées dans des systèmes hybrides (anomalies et mauvaises utilisations) avec un rappel de 91,25% (ligne 12) plutôt que pour des systèmes de détection des mauvaises utilisations où le rappel atteint seulement 53,25 % (ligne 10).

	Article	Dataset	Précision	Accuracy	Recall	Autres indicateurs	Commentaires
1	[12]	NSL-KDD avec tous (41) les attributs	/	97,94% [2]	/		
2	[12]	NSL-KDD avec certains (15) attributs sélectionnés avec CFS	/	98,88% [2]	/		
3	[9]	Partie de KDD'99	99,10%	93,77%	99,10%	AUC = 99,6% ; RMSE = 0,0682 ; FPRate = 0,1% ; FNRate = 6,1%	
4	[13]	New Kyoto 2006+ avec partie des attributs sélectionnés selon RF	99,00%	99,00%	99,00%		
5	[2]	NSL-KDD avec tous (41) les attributs	/	93,00%	89,10%	FPRate = 3,6% ; F-Measure = 0,923	
6	[2]	CICIDS2017	/	90,00%	92,60%	FPRate = 10,6% ; F-Measure = 0,785	
7	[4]	10% KDD'99 (8020)	/	/	50,73% [2]		
8	[4]	10% KDD'99 (8416)	/	/	53,00% [2]		
9	[4]	10% KDD'99 (8812)	/	/	51,63% [2]		
10	[7]	KDD'99 (mauvaise utilisation)	/	/	53,25% [2]		
11	[7]	NetFlow (anomalie)	/	/	65,00%	FPRate = 1 %	
12	[7]	KDD'99 (hybride)	/	/	91,25% [2]	FPRate = 1 %	

TABLE 2.4 – Performances des forêts aléatoires

2.4.4 K plus proches voisins

L'algorithme des K plus proches voisins (*K-Nearest Neighbor - KNN*) est un algorithme d'apprentissage basé sur les instances, c'est-à-dire qu'il ne contient pas d'étape de formation du modèle, il se contente de rechercher les exemples d'entrée et classe les nouvelles instances [10], [5]. KNN fait partie des algorithmes d'apprentissages paresseux non paramétriques. Le terme non paramétrique signifie qu'aucune hypothèse sur la distribution sous-jacente des données n'est effectuée. Tandis que paresseux signifie que la généralisation est retardée jusqu'au moment où la classification est effectuée. Cette méthode se base sur l'hypothèse que des instances de données existent à proximité d'autres instances de données qui ont des propriétés similaires. Une autre hypothèse est que les instances de données normales se trouvent dans des voisinages denses, tandis que les anomalies se trouvent loin de leurs voisins les plus proches [10]. Afin de mesurer la similarité ou non entre deux instances, l'utilisation d'une fonction de mesure de distance est envisagée afin d'évaluer cette différence ou similarité entre les instances. La mesure de distance

entre deux instances x et y la plus couramment utilisée étant la distance euclidienne :

$$d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2} \tag{2.13}$$

Où :
— x_k est le k^{ieme} attribut de l'instance x ;
— y_k est le k^{ieme} attribut de l'instance y.

Soit :
— S, le nombre total d'échantillons de l'ensemble de données ;
— $C = C_1, ..., C_L$, les L classes distinctes de l'ensemble S ;
— x, un vecteur d'entrée pour lequel la classe doit être prédite ;
— y_k, le k^{ieme} vecteur de l'ensemble S.

L'objectif de l'algorithme KNN est de trouver les K plus proches voisins de x dans S. Après cela, le vecteur x est classé dans la classe C_j si la majorité des K plus proches vecteurs appartiennent à la classe C_j [1], [5].

Le temps d'apprentissage est très limité mais la durée du processus de classification est plus importante étant donné les calculs à effectuer [10]. Le paramètre K a une grande importance dans le sens où ce paramètre peut influencer les performances. Si K est élevé, les voisins utilisés pour la prédiction nécessiteront un temps de classification important et auront une influence sur la précision de cette prédiction [5].

La particularité de la méthode KNN est qu'elle n'utilise pas les étiquettes des données. L'apprentissage peut être qualifié de non-supervisé [10].

La méthode des K plus proches voisins permet d'obtenir des résultats très satisfaisants dans l'ensemble. Cependant, pour la méthode dont les résultats sont présentés en ligne 3 du tableau 2.5, les attaques basses fréquences ne sont pas détectées. Cela signifie que les attaques de type U2R et R2L, qui sont peu nombreuses dans le jeu de données KDD'99, passeront entre les mailles du filet.

	Article	Dataset	Précision	Accuracy	Recall	Autres indicateurs	Commentaires
1	[3]	New Kyoto 2006+	95,65%	97,54%	91,67%		
2	[10]	KDD'99 avec 8 attributs sélectionnés avec chicharré	/	/	99,60%	FPRate : 0,1% ;	
3	[10]	KDD'99 avec 6 attributs sélection avec CANN [3] (approche basée sur la distance) (anomalie)	/	99,76%	99,99%	FPRate : 0,003% ;	Pas capable de détecter les attaques basses fréquences
4	[2]	NSL-KDD avec tous (41) les attributs	/	90,90%	90,50%	FPRate = 8,8% ; F-Measure = 0,903	
5	[2]	CICIDS2017	/	97,80%	98,60%	FPRate = 0,023% ; F-Measure = 0,944	
6	[1]	Partie de NSL-KDD (12,597 échantillons choisis aléatoirement)		99.6%			IPDS-KNN (Indexed Partial Distance Search KNN)
7	[1]	DARPA 1999		85,2%			K = 5
8	[16]	KDD'99	/	91,13%	76,17% [2]	FPRate = 12,03% [2] , F-Measure = 0,7771 [2]	
9	[16]	NSL-KDD	99,09% [2]	99,17%		FPRate = 0,26% [2], F-Measure = 0,9906 [2]	

TABLE 2.5 – Performances des K plus proches voisins

3. CANN est un système de détection des intrusions basé sur la combinaison des centres de regroupement et des voisins les plus proches.

2.4.5 Séparateur à vaste marge

Les séparateurs à vaste marge (SVM) ont été proposés la première fois par Vapnik en 1998 [5]. Le principe de cette méthode de classification se base sur la notion de marge [10] et sur la recherche d'un hyperplan séparateur dans l'espace des caractéristiques entre deux classes de manière à ce que la distance (appelée marge) entre l'hyperplan et les points des données les plus proches de chaque classe soit maximisée [7]. Les points qui se trouvent sur cette marge sont appelés les points du vecteur support et la solution est constituée sur base d'une combinaison linéaire de ceux-ci (figure 2.8). Cette approche est basée sur la minimisation du risque de classification plutôt que sur une classification optimale [7].
Le formalisme mathématique peut s'exprimer assez facilement dans un exemple à 2 dimensions [19]. Dans ce cas l'hyperplan est assimilé à une droite qui sépare le plan en 2 parties :

$$\langle w, x \rangle + b \geq 0 \tag{2.14}$$

$$\langle w, x \rangle + b < 0 \tag{2.15}$$

Les deux marges respectent les équations suivantes :

$$\langle w, x \rangle + b \pm 1 \tag{2.16}$$

Étant donné que la distance entre un point x_i et la droite $\langle w, x \rangle + b = 0$ vaut :

$$\frac{|\langle w, x_i \rangle + b|}{||w||} \tag{2.17}$$

La distance à maximiser entre les deux marges s'exprime comme :

$$M = \frac{2}{||w||} \tag{2.18}$$

Le problème se traduit donc par un problème d'optimisation quadratique, et plus précisément de minimisation, de la forme :

$$L(w) = \frac{||w||^2}{2} \tag{2.19}$$

sous les contraintes :

$$\langle w, x_i \rangle + b < 0 \longrightarrow y_i = -1 \tag{2.20}$$

$$\langle w, x_i \rangle + b > 0 \longrightarrow y_i = 1 \tag{2.21}$$

Lorsque les points ne sont pas linéairement séparables, les SVM utilisent des fonctions de noyaux ϕ appropriées pour transposer ces points dans des espaces de dimensions supérieures appelés espaces de caractéristiques (figure 2.7). L'hyperplan est alors défini dans ce nouvel espace [1], [10] sous la forme :

$$f(x) = \langle w, \phi(x) \rangle + b \tag{2.22}$$

Le choix des fonctions noyaux est très important. Il existe par exemple le noyau à base radiale (*Radial Basis Function - RBF*) qui peut être utilisé pour classifier des régions complexes [10]. D'autres noyaux existent comme les noyaux polynomiaux, gaussiens, ou tangents hyperboliques [7].

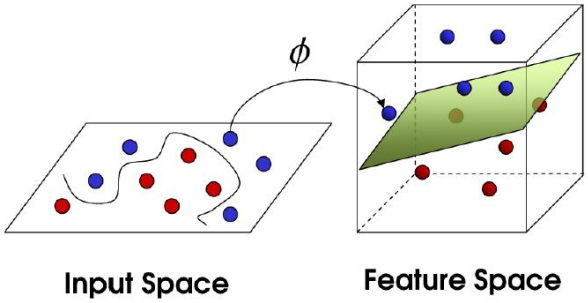

FIGURE 2.7 – Transformation des données dans un espace d'une autre dimension

Les SVM peuvent donc être divisés en 2 classes sur base de leur fonction noyau :
— SVM linéaire ;
— SVM non-linéaire.
Dans les SVM linéaires, les données d'entraînement sont séparées par l'hyperplan grâce à une fonction noyau linéaire. Lorsque les données sont plus complexes et ne sont pas linéairement séparables, l'utilisation de séparateur non linéaire donnera de mauvais résultats. C'est la raison pour laquelle on préfère, comme expliqué plus haut, transposer les données dans des espaces de caractéristiques de dimensions plus hautes afin d'y trouver un hyperplan linéaire [10].

Selon le type de détection, anomalies ou mauvaises utilisations, les SVM peuvent être séparés en tant que SVM multi-classes ou SVM mono-classe. La classification multi-classe est utilisée lorsque les données sont labellisées pour réaliser un apprentissage supervisé. Elle peut être effectuée par deux méthodes : *one-versus-all* qui est la méthode classique, le problème de classification multi-classe est alors séparé en un problème de classification binaire pour chaque classe, et *one-versus-one*, dans laquelle un classifieur SVM binaire est conçu pour chaque paire de classes et la classe sélectionnée est celle qui est obtenue par la majorité des classifieurs [7], [10]. Les classifieurs SVM mono-classe sont utilisés pour de l'apprentissage non-supervisé, que les nouvelles données soient similaires ou différentes des données d'entraînement [10].

Les SVM ont une très bonne capacité de généralisation et sont très utiles lorsque le nombre de caractéristiques est élevé et que le nombre de points de données est faible [7]. Cependant, l'exécution des méthodes SVM demande une capacité de calcul importante et la complexité algorithmique est en $O(n^2)$ où n représente le nombre d'instances [11], ce qui est relativement élevé. Les performances dépendent du choix des fonctions noyaux et des paramètres de ces fonctions. La durée d'entraînement est également élevée mais les SVM sont des méthodes qui résistent aux bruits dans les données [10].

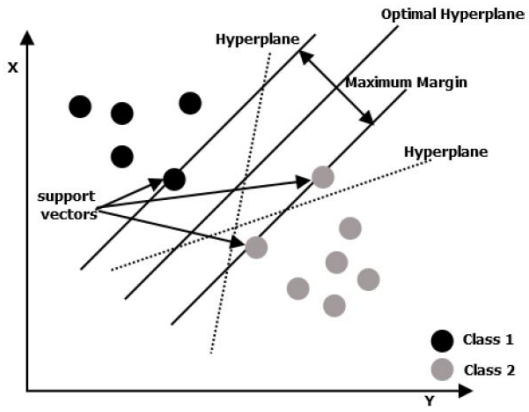

FIGURE 2.8 – Exemple de SVM [10]

A propos des résultats obtenus en intégrant les séparateurs à vaste marge dans des systèmes de détection d'intrusions, le constat qui peut être fait rapidement, au regard des lignes 5 à 8, est que les systèmes hybrides sont moins performants que les systèmes basés sur la détection des anomalies. En effet, que ce soit avec le jeu de de données NSL-KDD ou DAPRA, l'accuracy obtenue est meilleure (89,70% et 95,11% respectivement) pour la détection des anomalies que pour la détection hybride (82,37% et 69,80% respectivement). La comparaison des lignes 1 et 2, montre également que l'utilisation du noyau de fonction de base radiale , permet d'atteindre de meilleures performances lorsqu'il s'agit du jeu de données New Kyoto 2006+. En effet, lors de son utilisation, la précision est de 92%, l'accuracy de 97,54% et le rappel de 95,83% alors que dans le cas contraire, ces mêmes indicateurs valent 86,95%, 94,26% et 83,33% respectivement.

	Article	Dataset	Précision	Accuracy	Rappel	Autres indicateurs	Commentaires
1	[3]	New Kyoto 2006+	86,95%	94,26%	83,33%		
2	[3]	New Kyoto 2006+	92,00%	97,54%	95,83%		Noyau RBF
3	[12]	NSL-KDD avec tous (41) les attributs	/	94,42% [2]	/		
4	[12]	NSL-KDD avec certains (15) attributs sélectionnés avec CFS	/	95,72% [2]	/		
5	[11]	NSL-KDD (anomalies)	/	89,70%	/		
6	[11]	NSL-KDD (hybride)	74,00%	82,37%	82,00%		
7	[11]	DAPRA (hybride)	/	69,80%	/		
8	[11]	DAPRA (anomalies)	/	95,11%	/		
9	[6]	Partie de KDD'99	/	99,50%	/		
10	[7]	Partie de KDD'99 avec 19 attributs sélectionnés avec une politique de suppression de caractéristiques et une politique de sélection d'une seule caractéristique (mauvaise utilisation)	/	98,00%	/		Noyau RBF - Sous ensemble sélectionné avec optimisation par colonie de fourmis (maximiser généralisation et minimiser le biais dans KDD'99)
11	[7]	Partie de KDD'99 avec 19 attributs sélectionnés avec différentes méthodes de sélection (caractéristiques qui maximisent la performance, basée sur information mutuelle et basée sur corrélation) (mauvaise utilisation)	/	/	97,80% [2]		KDD'99 échantillonné pour avoir 7000 échantillons de chaque catégorie d'attaques, normaux y compris
12	[7]	Partie de DAPRA 1998 (anomalie)	/	100,00%	/	FPRate = 3%	
13	[7]	NetFlow collecté à partir de données réelles et simulées par l'outil Flame (anomalies)	/	94,00%	/	FPRate = 3%	
14	[13]	New Kyoto 2006+ avec partie des attributs sélectionnés selon RF	95,00%	95,00%	95,00%		
15	[10]	KDD'99 avec tous les attributs (mauvaise utilisation)	/	/	40,56% [2]		
16	[10]	DAPRA 1998 avec tous les attributs (hybride)	/	69,80%	62,44% [2]	FPRate = 37.8% ; FNRate = 29.8%	
17	[10]	NSL-KDD	/	99,31%	99,2%	FPRate = 0,6%	La méthode de transformation du logarithme de la densité marginale (*Logarithm marginal density ratios transformation - LMDRT*) est utilisée pour effectuer la transformation des données.
18	[10]	KDD'99 avec certains (17-24) attributs sélectionnés avec la méthode de suppression progressive des caractéristiques (*Gradual Feature Removal - GFR*)(hybride)	/	95,70%	61,33% [2]	FPRate = 0.7%	Noyau RBF
19	[10]	KDD'99 sans répétition avec certains (19) attributs sélectionnés avec GFR (hybride)	/	98,62%	83,33% [2]		Sous ensemble d'entraînements sélectionnés avec les colonnies de fourmis
20	[10]	KDD'99 avec certains attributs sélectionnés avec les algorithmes génétiques (mauvaise utilisation)	/	/	92,27%	FPRate = 1,025%	Données sont prétraitées avec analyse en composantes principales pour réduire les attributs qui appartiennent à des espaces de haute dimension
21	[2]	NSL-KDD avec tous (41) les attributs	/	89,70%	81,90%	FPRate = 3,5% ; F-Measure = 0,881	
22	[2]	CICIDS2017	/	96,80%	92,50%	FPRate = 2,3% ; F-Measure = 0,912	
23	[1]	10% KDD Cup 99		99,9%			Noyau RBF
24	[1]	KDD'99 avec 18 attributs sélectionnés avec le gain d'information et optimisation binaire par essaims de particules (*Binary Particle-Swarm Optimization - BSPO*)	84,2%	99,0%			
25	[1]	NSL-KDD	74%	82,37%			
26	[1]	Partie de DARPA 1998 avec 18 attributs	81,2%	80,1%			
27	[1]	DARPA 1998		95,11%			Algorithme qui se concentre sur les attaques DDOS
28	[16]	KDD'99	/	94,46%	69,49% [2]	FPRate = 8,22% [2] , F-Measure = 0,6287 [2]	
29	[16]	NSL-KDD	62,46% [2]	66,02%	/	FPRate = 7,53% [2], F-Measure = 0,5824 [2]	

TABLE 2.6 – Performances des séparateurs à vaste marge

2.4.6 Réseau de neurones artificiels

Les réseaux de neurones artificiels (*Artificial Neural Network - ANN*) se sont inspirés du cerveau humain et sont composés de neurones artificiels interconnectés les uns avec les autres et capables d'effectuer certaines opérations sur les données qui leurs sont soumises [7]. Ces réseaux de neurones artificiels sont constitués de trois éléments importants :

— Noeuds d'entrée ;
— Noeuds cachés ;
— Noeuds de sortie.

Les données d'entrée activent les neurones de la première couche cachée, les sorties de cette première couche vont ensuite servir d'entrée à la deuxième couche cachée et ainsi de suite jusqu'à la couche de sortie qui nous permet d'obtenir un résultat de classification (figure 2.9) [7].

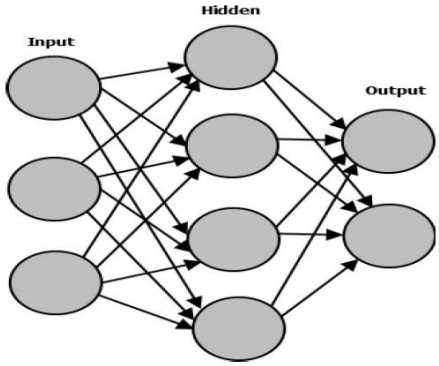

FIGURE 2.9 – Exemple de réseau de neurones avec une seule couche cachée [10]

Il existe un grand nombre d'algorithmes basés sur les réseaux de neurones artificiels dont les plus connus et les plus utilisés sont ceux basés sur les perceptrons multicouches [10]. Pour les réseaux de ce type, l'entraînement est généralement réalisé en utilisant l'algorithme de retropropagation. Au départ, les connexions entre les neurones possèdent des poids aléatoires. Ensuite, au cours des différentes itérations, l'algorithme ajuste les poids afin de minimiser l'erreur de classification [5], [9]. Les réseaux de neurones entraînés avec l'algorithme de retropropagation sont constitués de deux étapes :

— Propagation avant (*feed forward*) ;
— Retropropagation (*backpropagation*).

Comme expliqué plus haut, la propagation avant est l'étape où les données d'entrée sont transmises à chaque noeud d'une couche et chacun de ces noeuds, cachés ou de sortie, calcule sa valeur d'activation obtenue grâce à la fonction d'activation. La fonction d'activation, représentée par une fonction mathématique, est une sorte de porte logique située entre l'entrée du neurone et sa sortie, elle-même reliée à la couche suivante. Elle décide si le neurone doit être activé ou non. La différence entre la valeur de sortie réelle et la valeur de sortie souhaitée est utilisée pour calculer une erreur. Dans l'étape de retropropagation, l'erreur est propagée à partir de la couche de sortie jusqu'à la couche d'entrée. Les poids sont ainsi mis à jour entre les noeuds de sortie et les noeuds cachés. La méthode de descente du gradient est généralement utilisée pour effectuer cette mise à jour des poids. La mise à jour s'arrête lorsqu'un seuil prédéfini est atteint [10].

Les réseaux de neurones artificiels ont la capacité d'apprendre par l'exemple et de pouvoir généraliser à partir de données qui sont limitées en quantité, bruitées et incomplètes [6]. Cependant, contrairement aux séparateurs à vaste marge, (section 2.4.5), les réseaux de neurones artificiels sont sensibles au nombre de caractéristiques présentes dans les données. En effet, si ce nombre de caractéristiques augmente, le temps d'apprentissage augmente également. Les ANN ont la capacité de créer des modèles de classification non linéaires, qu'ils contiennent une ou plusieurs couches cachées. Les versions avancées de ces réseaux de neurones nécessitent également une capacité de calcul importante, c'est pourquoi ils sont souvent implémentés sur des ressources graphiques [7]. Les réseaux neuronaux entraînés avec l'algorithme de retropropagation peuvent atteindre facilement un minimum local, ce qui engendre une stabilité plus faible [10]. Une étape difficile est de trouver le nombre optimal de couches cachées ainsi que le nombre de neurones qui constituent ces couches [10], [9]. La fonction d'activation (par exemple sigmoïde sur la figure 2.10, unité rectifiée linéaire (ReLU) sur la figure 2.12 ou tangente hyperbolique sur la figure 2.11) a aussi une influence sur les performances. Un dernier inconvénient des réseaux de neurones artificiels est que la fonction d'apprentissage est difficilement interprétable par l'Homme [10].

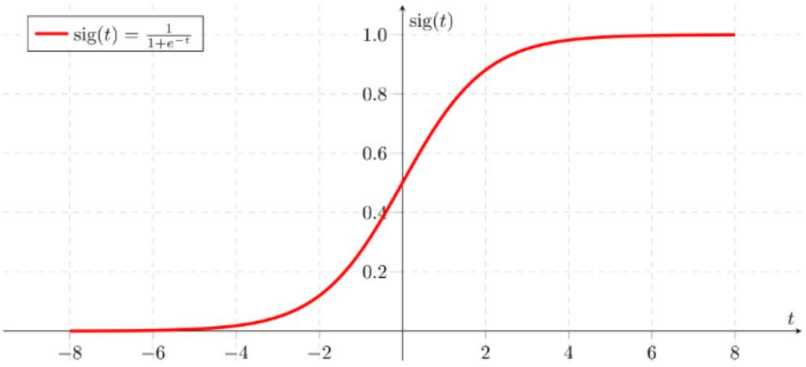

FIGURE 2.10 – Fonction d'activation sigmoïde [20]

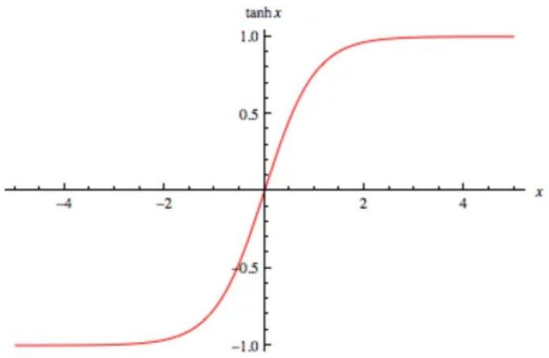

FIGURE 2.11 – Fonction d'activation tangente hyperbolique [20]

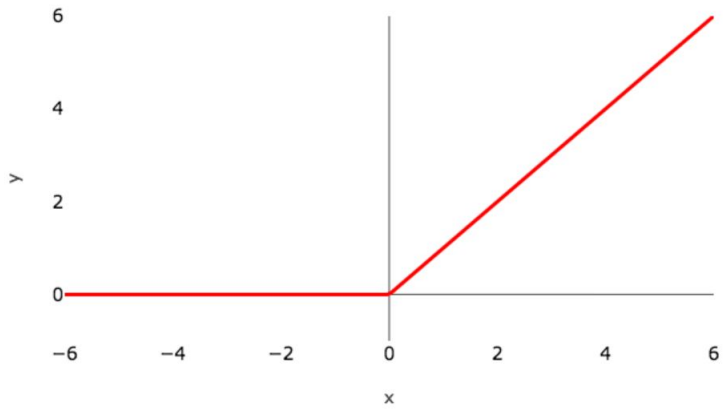

FIGURE 2.12 – Fonction d'activation ReLU [20]

Concernant l'utilisation des réseaux de neurones artificiels pour la détection d'attaques, il existe certaines limitations [10] :

— La précision de détection est faible surtout pour les attaques basse fréquence ;
— Ils ne peuvent pas détecter les attaques temporairement dispersées et collaboratives car ils sont incapables de retenir des évènements passés. Cette limitation s'applique uniquement aux réseaux de neurones "classiques". En effet, les réseaux de neurones LSTM (*Long Short-Term Memory*) ont, quant à eux, des capacités à retenir des évènements passés.

De manière similaire aux constatations effectuées pour les classifieurs bayésiens naïfs (section 2.4.1) et pour les forêts aléatoires (section 2.4.3), les résultats obtenus en limitant le nombre d'attributs des données, dans ce cas-ci sur le jeu de données KDD'99, sont meilleurs que si tous les attributs sont conservés.

En effet, les indicateurs à la ligne 3 du tableau 2.7 témoignent des performances lorsque les attributs sont sélectionnés sur base de la corrélation (CFS), ils sont meilleurs que pour la ligne 2 où tous les attributs sont conservés et qui présente notamment un taux de faux positifs de 0,4%, c'est-à-dire 4 fois plus important. En parlant du taux de faux positifs, la ligne 7 fait état d'une valeur de 76%, ce qui est très mauvais. Cette ligne concerne un système qui permet de détecter les anomalies et une comparaison avec un système de détection des mauvaises utilisations n'est malheureusement pas envisageable étant donné l'absence de résultat pour cette option. Le perceptron multicouche (*Multilayer Perceptron - MLP*) semble largement utilisé comme réseau de neurones artificiels, cependant les résultats paraissent assez variables en comparant les lignes 1,2 et 3 avec les lignes 10 et 11 où les performances sont inférieures.

	Article	Dataset	Précision	Accuracy	Rappel	Autres indicateurs	Commentaires
1	[9]	Partie de KDD'99	97,80%	91,90%	97,80%	AUC = 0,99 ; RMSE = 0,0813 ; FPRate = 1,4% ; FNRate = 6,6%	MLP
2	[8]	10% KDD'99 avec tous les attributs	97,70%	98,75%	98,80%	AUC = 0,998 ; FPRate = 0,4% ; F-measure = 0,982 ; Kappa Statistic = 0,979 ; MAE = 0,0011	MLP
3	[8]	KDD'99 avec 11 attributs sélectionnés avec CFS	99,30%	99,28%	99,30%	AUC = 0,999 ; FPRate = 0,1% ; F-measure = 0,991 ; Kappa Statistic = 0,988 ; MAE = 0,001	MLP
4	[6]	Partie de KDD'99	/	99,25%	/		
5	[7]	*Network packet-level data* (mauvaise utilisation)		93,00%			
6	[7]	DAPRA 1998 (anomalie)	/	/	80,00%		
7	[7]	DAPRA 1999 (anomalie)	/	/	/	FPRate = 76% ; Prediction Rate Normal = 100%	
8	[13]	New Kyoto 2006+ avec partie des attributs sélectionnés selon RF	98,00%	98,00%	98,00%		
9	[10]	KDD'99 avec tous les attributs	/	91,90%	51,51%		Taille entrée = 41 - Taille cachée = 29 - Taille sortie = 5
10	[16]	KDD'99	/	86,53%	60,20% [2]	FPRate = 5,2% [2], F-Measure = 0,3905 [2]	MLP
11	[16]	NSL-KDD	49,96% [2]	71,72%	/	FPRate = 7,72% [2], F-Measure = 0,4323 [2]	MLP

TABLE 2.7 – Performances des réseaux de neurones artificiels

2.4.7 Logique floue

La logique floue, aussi appelée théorie des ensembles flous, se base sur le concept de phénomènes flous qui se produisent régulièrement dans le monde réel [5]. Cette théorie prend en compte le degré de vérité plutôt que la logique booléenne (vrai ou faux). Grâce à cette théorie, les objets peuvent appartenir à plusieurs classes en même temps [5], [6], [10]. Cet élément constitue un avantage considérable pour la détection des intrusions. Effectivement, dans la sécurité informatique, la délimitation entre le normal et l'anomalie n'est pas bien définie. Les données utilisées pour la détection des intrusions contiennent de nombreux attributs numériques mais aussi des mesures statistiques dérivées. La construction de modèles à partir de ces données entraîne, la plupart du temps, un taux d'erreur de détection élevé. Cela s'explique par le fait qu'un comportement qui dévie légèrement ne sera peut-être pas détecté, tout comme de petits changements par rapport à un comportement normal pourront être interprétés comme une anomalie. La logique floue permet de modéliser ces faibles dérives afin de conserver des taux de faux positifs et de faux négatifs faibles [6].

Chaque règle de la logique floue peut s'exprimer de la manière suivante [6] :

```
SI condition ALORS conclusion [poids]
```

Où :
- Condition est une expression floue décrite par des opérateurs logiques flous comme le ET flou ou le OU flou ;
- Conclusion est une expression atomique ;
- Poids est un nombre réel compris entre 0 et 1.

Un ensemble flou A dans X est caractérisé par une fonction d'appartenance $fA(x)$ qui, à chaque point de X, associe un nombre réel dans [0 ; 1]. Les valeurs de $fA(x)$ en x représentant le "degré d'appartenance" de x à A. Cela signifie que plus $fA(x)$ est proche de 1, plus le degré d'appartenance de x à A est élevé [10]. Un exemple de fonction d'appartenance est illustré sur

la figure 2.13.

Utilisée seule, la logique floue n'est pas suffisante pour détecter les attaques de tous les types. Pour atteindre des performances satisfaisantes, elle est intégrée à d'autres types de classifieurs. La construction d'un modèle flou est assez complexe par rapport à d'autres solutions d'apprentissage automatique, cette complexité est notamment due aux réglages fins et au nombre important de simulations qui doivent être effectués avant que le modèle ne soit opérationnel [10].

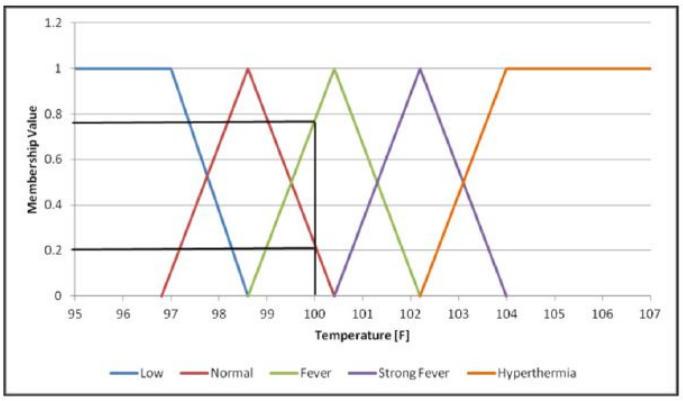

FIGURE 2.13 – Exemple de fonction d'appartenance pour la variable floue : Température du corps humain [7]

Le nombre de résultats pour les méthodes utilisant la logique floue est très limité, comme en témoigne le tableau 2.8. La ligne 1 permet de constater que la méthode atteint une *accuracy* de 100% mais le taux de faux positifs est relativement élevé (13%). Cela signifie que si une personne doit analyser toutes les instances qui sont identifiées comme étant des intrusions, de nombreuses instances, qui sont en réalité tout à fait classiques, seront transmises pour analyse inutilement.

	Article	Dataset	Précision	Accuracy	Rappel	Autres indicateurs	Commentaires
1	[7]	KDD'99 version corrigée avec 300,000 instances (hybride)		100,00%		FPRate = 13%	
2	[10]	KDD'99 avec tous les attributs			60,55% [2]	FPRate = 2%	

TABLE 2.8 – Performances de la logique floue

2.4.8 Algorithme génétique

Les algorithmes génétiques (*Genetical Algorithms - GA*) font partie d'une famille de méthodes basées sur le calcul évolutionnaire qui comprend aussi la programmation génétique (*Genetical Programming - GP*), l'optimisation par essaim de particules, l'optimisation par colonies de fourmis et les systèmes immunitaires artificiels [7]. Les algorithmes génétiques utilisent l'ordinateur pour simuler la sélection naturelle (survie du plus apte) et l'évolution [5], [7]. Le principe des algorithmes génétiques est que chaque solution possible d'un problème est représentée comme une séquence de bits (un bit = un gène) qui est appelée génome ou chromosome [6], [10]. Un algorithme commence donc avec un ensemble de génomes choisis aléatoirement qui constitue une population. Une fonction *fitness* est ensuite utilisée pour évaluer la performance

de chaque individu (génome) dans une population. Ensuite, chaque individu subit des transformations à l'aide d'opérateurs de reproduction qui peuvent être une mutation ou un croisement [5], [6], [7], [10]. Un croisement détermine la manière dont les propriétés des parents vont être transmises aux enfants, tandis que la mutation est une altération spontanée d'un seul gène [6], [7]. Les individus qui obtiennent des bons résultats avec la fonction de *fitness* ont plus de chance d'être choisis pour subir l'une ou l'autre modification. Les individus les mieux adaptés sont donc répliqués dans la génération suivante et les individus peu performants sont remplacés par des recombinaisons d'individus performants. [7], [5]. Lorsque l'algorithme génétique a atteint un certain nombre de générations, il s'arrête et les meilleurs individus sont sélectionnés [6]. La principale différence entre les algorithmes génétiques et la programmation génétique étant que, dans le premier cas, les individus sont représentés par des chaînes binaires tandis que pour la programmation génétique, les individus sont représentés des programmes donc des arbres avec des opérateurs (plus, moins, multiplier, diviser, ou, et, pas) ou même des blocs de programmation (si alors, boucle, etc.). Dans la programmation génétique, les opérateurs de croisement (*crossover*) et de mutation sont beaucoup plus complexes que ceux utilisés dans les algorithmes génétiques [7].

Pour l'application à la détection des intrusions, les éléments importants sont la vitesse, la précision et l'adaptabilité. Les algorithmes génétiques sont performants quand ils sont appliqués avec d'autres classifieurs pour en optimiser les paramètres, pour faire évoluer les règles dans le cas des arbres de décision [6] comme ceux proposés par Sinclair, Pierce et Matzner [21] ou encore pour sélectionner les attributs des données. Par contre, rien ne garantit que l'algorithme génétique arrive à trouver un optimum global [10].

Les indicateurs sont peu nombreux pour les algorithmes génétiques et la programmation génétique, une comparaison est donc difficilement réalisable sur base du contenu du tableau 2.9. Cependant, sur base du taux de faux positifs, la programmation génétique (lignes 1,2 et 4) semble donner de meilleurs résultats que les algorithmes génétiques (ligne 3). Bien que les algorithmes génétiques obtiennent un rappel correct de 93,78%.

	Article	Dataset	Précision	Accuracy	Rappel	Autres indicateurs	Commentaires
1	[7]	DAPRA 1998 avec différents ensembles d'attributs en fonction des différentes techniques de programmation génétique	/	/	/	FPRate = 0 - 5,7%	GP
2	[7]	Partie de KDD'99 (30,000 instances entraînement et 10,000 pour test)	/	/	/	FPRate très faible	GP
3	[7]	Partie de KDD'99 (10,000 instances entraînement et 10,000 pour test)	/	/	93,78%	FPRate = 2,75 - 10,8%	GA
4	[7]	Partie de DAPRA 1999 (10,000 instances entraînement et 10,000 pour test)	/	/	/	FPRate = 0,0041	GP

Table 2.9 – Performances des algorithmes génétiques

2.4.9 Réseaux bayésiens

Les réseaux bayésiens sont des modèles graphiques probabilistes qui représentent les variables qui, dans notre cas, sont les attributs, ainsi que les relations entre ces variables. Le réseau est construit avec des noeuds qui sont associés aux variables qui peuvent être discrètes ou continues et des arêtes orientées qui font état des relations entre les variables, ce qui donne un graphe acyclique orienté. Les noeuds enfants sont dépendants des noeuds parents. Chaque

noeud conserve les états des variables aléatoires et la forme des probabilités conditionnelles. Les réseaux bayésiens sont établis grâce à des experts ou via des algorithmes efficaces qui effectuent des inférences [7]. Un exemple de réseau bayésien pour la détection de signature est représenté sur la figure 2.14.

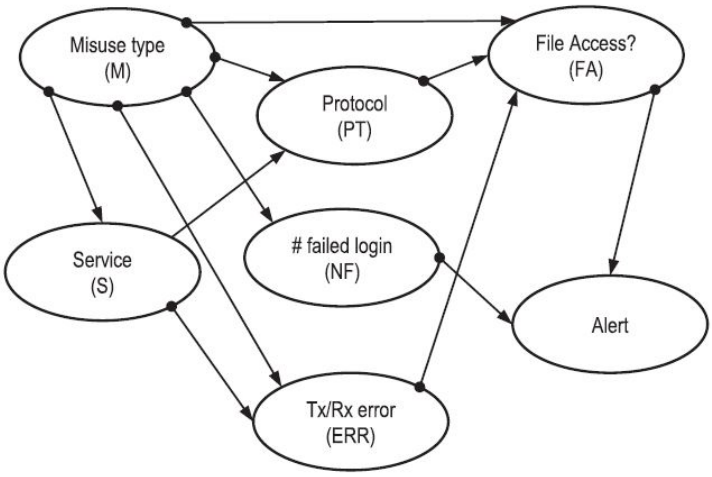

FIGURE 2.14 – Exemple de réseau bayésien pour la détection de signature [7]

Ce schéma doit être complété par un ensemble de probabilités conditionnelles qui peuvent prendre, par exemple, les formes suivantes :

$$P(FA = True|M = R2H, PT = NSF, ERR = 0) = 0,95 \tag{2.23}$$

$$P(FA = True|M = Probe, PT = none, ERR = 50\%) = 0,80 \tag{2.24}$$

$$P(FA = True|M = DoS, PT = HTTP, ERR = 50\%) = 0,90 \tag{2.25}$$

Les lignes 2 et 3 du tableau 2.10, rapportant les différents résultats des systèmes de détection d'intrusion utilisant les réseaux bayésiens, montrent que la limitation du nombre d'attributs pour le jeu de données KDD'99 n'a pas d'influence significative sur les performances. Les deux scénarios présentent effectivement de très bons résultats autant en termes de précision, de rappel, autour des 99,7 % dans les deux cas, que de AUC (0,997 et 1) qui traduit des classifieurs fleurtant avec le classifieur idéal.

	Article	Dataset	Precision	Accuracy	Rappel	Autres indicateurs	Commentaires
1	[9]	Partie de KDD'99	99,20%	90,73%	99,20%	AUC = 0,997 ; RMSE = 0,087 ; FPRate = 0,1% ; FNRate = 9,2%	
2	[8]	10% KDD'99 avec tous les attributs	99,80%	99,67%	99,70%	AUC = 1 ; FPRate = 0% ; F-measure = 0,997 ; Kappa Statistic = 0,9944 ; MAE = 0,0003	
3	[8]	KDD'99 avec 11 attributs sélectionnés avec CFS	99,80%	99,72%	99,70%	AUC = 0,997 ; FPRate = 0% ; F-measure = 0,997 ; Kappa Statistic = 0,9952 ; MAE = 0,0003	
4	[7]	*tcpdump - botnet traffic* (mauvaise utilisation)	93,00%	/	/	FPRate = 1,39%	
5	[7]	KDD'99 avec 9 attributs (mauvaise utilisation)	/	/	60,80% [2]		
6	[7]	DAPRA 1999 avec certains attributs (anomalie)	/	75 - 100%	/	FPRate = 0,1 - 0,2 %	

TABLE 2.10 – Performances des réseaux bayésiens

2.4.10 Modèle de Markov caché

Un modèle de Markov est un modèle composé d'états de transitions et d'actions. Les modèles de Markov comprennent les chaînes de Markov et les modèles de Markov cachés (*Hidden Markov Models - HMM*) qui nous intéressent ici [7]. Une chaîne de Markov est constituée d'un ensemble d'états interconnectés qui déterminent la topologie du modèle. Pour les modèles de Markov cachés, les systèmes modélisés sont représentés par des processus de Markov dont les paramètres sont inconnus [7], [10]. Le défi principal est de définir les paramètres cachés à partir des paramètres observables. Les états d'un HMM représentent les conditions non observables modélisées [7]. Dans un HMM, une observation X_t au temps t est générée via un processus stochastique. Mais, l'état Z_t du processus ne peut pas être directement observé car il est caché. Les modèles de Markov cachés satisfont aux propriétés des modèles de Markov qui sont que l'état Z_t dépend uniquement de l'état Z_{t-1} [7]. Les probabilités de transitions entre les états sont définies via la matrice de transition P qui est une matrice carrée dont chaque élément A_{ij} décrit la probabilité de passer de l'état i à l'état j du temps $t-1$ au temps t. Cela peut se résumer sous la forme : $A_{ij} = P(Z_{t,j}|Z_{t-1,i})$ [10]. La matrice Q, qui est la matrice de probabilité d'observation, définit les probabilités de recevoir différentes observations, étant donné que l'hôte est dans un certain état. π est la distribution de l'état initial. Un HMM est désigné par (P, Q, π) [7].

Les HMM ont l'avantage de bien saisir les dépendances entre les séquences consécutives. Ils sont également performants pour reconnaître la structure d'une séquence. Mais, ces modèles présentent tout de même un inconvénient. Lorsqu'ils sont entièrement connectés, ils peuvent être soumis au sur-apprentissage. Ce problème arrive quand le modèle est entraîné avec un ensemble de données qui ont un grand espace de paramètres [10].

Le tableau 2.11 reprend les performances des systèmes de détection d'intrusions qui font usage des modèles de Markov cachés. Le nombre de ces résultats est faible ce qui rend des comparaisons assez difficiles. Cependant, la ligne 1 montre de bons résultats avec une AUC variant entre 0,915 et 0,976 et un taux de faux positifs très bas (0,001%) contrairement à ligne 2 où ce taux est de 21%.

	Article	Dataset	Precision	Accuracy	Rappel	Autres indicateurs	Commentaires
1	[7]	*HTTP payload* (mauvaise utilisation)	/	/	85,00%	AUC = 0,915 - 0,976 ; FPRate = 0,001%	
2	[7]	KDD'99 avec 5 attributs sélectionnés (anomalie)	/	/	79,00%	FPRate = 21%	

TABLE 2.11 – Performances des modèles de Markov cachés

2.4.11 Apprentissage ensembliste

Les classifieurs d'ensemble ont été proposés afin d'améliorer les performances obtenues à l'aide d'un seul classifieur [22] [5]. L'apprentissage ensembliste combine les prédictions de plusieurs apprenants faibles. Cette combinaison permet de générer un ensemble d'hypothèses pour un même problème dans le but de former une meilleure hypothèse que la meilleure hypothèse seule [7], [5], [10]. Un apprenant faible étant un apprenant qui permet d'obtenir de meilleurs résultats que le hasard [7].

Il existe plusieurs méthodes de combinaison [5], [7], [10] :

— Le vote majoritaire ;
— Le *boosting* ;
— Le *bagging* ;
— L'empilement.

Le vote majoritaire est une méthode d'ensemble dont la classe prédite résulte tout simplement de la classe qui a reçu le plus de votes parmi les classifieurs constituant l'ensemble.

Le *boosting* consiste à former un ensemble de données sans replacement, c'est-à-dire que toute observation ne peut pas être répétée dans un sous-ensemble de données. Cet ensemble est utilisé pour former un apprenant faible. Dans l'étape suivante, on extrait un sous-ensemble de données de la même manière que précédemment, à l'exception près que 50% des données mal classées à l'itération précédente sont ajoutées à ce nouvel ensemble et un nouvel apprenant faible est formé. A la troisième itération, on crée un nouvel ensemble d'apprentissage avec les données qui ont été mal classées en utilisant les deux apprenants faibles précédents et ainsi de suite [10]. Lorsqu'on a atteint un certain nombre d'itérations, les différentes prédictions obtenues par les apprenants faibles sont combinées en utilisant le schéma du vote majoritaire dans le cas d'une classification ou via la règle de la moyenne dans le cas d'une régression [10], [5]. En résumé, le *boosting* est une sorte de régression linéaire où l'entrée de l'apprenant faible h (ex. une ligne droite divisant les données en deux catégories) sont les attributs des données et la sortie est une somme pondérée de ces h fonctions [7]. L'algorithme le plus connu utilisant le *boosting* est l'algorithme AdaBoost [7], [16].

Le *bagging*, aussi appelé agrégation bootstrap, consiste à créer des échantillons bootstrap. Un échantillon bootstrap est constitué en sélectionnant un échantillon de données à partir de l'ensemble des données d'apprentissage. Les nouveaux sous-ensembles d'apprentissage sont créés via un échantillonage avec replacement, une observation peut donc être répétée plusieurs fois dans un même ensemble. Chacun de ces ensembles est utilisé pour créer un modèle (apprenant faible) et les sorties de ces apprenants faibles sont combinées, comme pour le *boosting*, par vote majoritaire (classification) ou par moyenne (régression) [10]. Le *bagging* vise à améliorer l'aptitude de généralisation des modèles prédictifs en réduisant le sur-apprentissage [7]. Il est également utilisé pour réduire la variance des algorithmes d'apprentissage. Cette variance peut être définie comme la quantité de changements dans la prédiction de la fonction cible pour les différents ensembles de données d'apprentissage [10].

L'empilement est un méta-algorithme qui rassemble n classifieurs/apprenants faibles pour l'ensemble des données d'apprentissage. Chaque classifieur génère des prédictions sur base de ces données. La sortie de chacun de ces classifieurs est ensuite de nouveau apprise par un combinateur ou un méta-algorithme qui effectue une prédiction finale [10].

Une méthode d'apprentissage d'ensemble connue et très employée est la méthode des forêts aléatoires qui est expliquée dans la section 2.4.3.

L'apprentissage ensembliste comporte certains inconvénients comme la complexité de l'ensemble qui affecte le temps d'apprentissage. Il nécessite également plus de mémoire que les classifieurs uniques [10]. Cependant, l'apprentissage ensembliste permet d'améliorer la capacité de généralisation des algorithmes d'apprentissage, comme déjà mentionné plus haut. Il permet également de réduire le sur-apprentissage et de diminuer les erreurs de prédiction [7], [10].

Comme le mentionnent de nombreux articles ([3]), [4], [8], [23], [6], [7]), les méthodes ensemblistes permettent d'obtenir de meilleurs résultats et le tableau 2.12 confirme cette tendance car les performances observées sont très bonnes mis à part quelques indicateurs plus mitigés pour les lignes 5,11 et 13. La méthode ensembliste AdaBoost M1, qui est une implémentation particulière du *boosting*, témoigne de ces bons résultats avec une *accuracy* qui approche les 100%, un taux de faux positifs nul ou presque nul en fonction du jeu de données utilisé (KDD'99 ou NSL-KDD). Dans les deux cas, les attributs ont été sélectionnés avec la méthode basée sur la corrélation entre eux.

	Article	Dataset	Precision	Accuracy	Rappel	Autres indicateurs	Commentaires
1	[3]	New Kyoto 2006+	88,46%	96,72%	95,83%		Vote majoritaire avec Kmeans, KNN, Fuzzy C-Means, Bayes Naïf, SVM et SVM RBF
2	[10]	KDD'99 avec tous les attributs (hybride)	91,30%	/	99,08%	F-Measure = 0,9504	Création de différents ensembles d'entraînement en utilisant la classification floues (Fuzzy C-Means) et entrainement d'un réseau de neurones sur chaque ensemble puis combinaison des résultats pour entrainer un dernier réseau de neurones pour classification
3	[6]	KDD'99	/	/	98,95%	FPRate = 7%	Algorithme génétique pour générer des classifieurs flous
4	[7], [10]	Partie de DAPRA 1998 (80% attaques et 20% normal) (mauvaise utilisation)	/	99,82%	95,11% [2]		Vote majoritaire avec trois ANN différents + un spline de régression adaptative multivariée (*Multivariate adaptive regression spline - MARS*) + un SVM
5	[10]	KDD'99 (hybride)	/	97,79% [2]	48,96% [2]		Pré-traitement des données avec K-means clustering, sélection de différents ensembles d'attributs avec réseau de neurones flous (*Neuro-Fuzzy*), classification avec SVM sur base des différents ensembles d'attributs
6	[10]	KDD'99 (hybride)	/	/	95.30%	FPRate = 4,25% ; FNRate = 3%	SVM entrainé à partir des sous ensembles des données puis optimisation des vecteurs de support avec CSOACN (*Clustering based Self-Organized Ant Colony Networks*) pour entraînement suivant des SVM. Au final, 2 modèles entrainés CSOACN et SVM et si les deux modèles ne sont pas d'accord données classifiées comme "amphibies"
7	[10]	KDD'99	/	92,90%	/		Six SVM + six KNN et combinaison des résultats selon différentes méthodes : vote majoritaire pondéré avec différentes méthodes pour calculer les poids
8	[1]	10% KDD'99	99,5%	98,9%	/		Fuzzy C-Means pour regrouper les données puis ANN entraîné sur les différents sous-ensembles des données et SVM pour classification finale
9	[1]	KDD'99	/	99.89%	/	FPRate = 0,11%	Système basé sur arbres de décision (C4.5) et algorithmes génétiques
10	[10]	KDD'99 avec tous les attributs (hybride)	/	/	99,25% [3]		Données passent dans un premier classifieur RBF NN (*Radial Basis Function Neural Network*) et puis passent dans un des 4 classifieurs entraînés séparément sur chacune des catégories d'attaques.
11	[10]	DAPRA 1998 avec tous les attributs (hybride)	/	69,80%	62,44% [2]	FPRate = 37,8% ; FNRate = 29,8%	Classification hiérarchique de type arbre de décision avec ajout et optimisation des nœuds avec SVM
12	[10]	DAPRA avec sélection des attributs avec les algorithmes génétiques (mauvaise utilisation)	/	/	98,98% [2]		Arbres neuronaux flexibles (*Flexible Neural Tree - FNT*) avec optimisation des paramètres par optimisation des essaims

13	[10]	KDD'99 (hybride)	/	/	77,61% [2]		Empilement arbres de décision (C4.5), classification bayésienne, C4.5
14	[10]	DAPRA avec tous les attributs (mauvaise utilisation)	/	/	93,75% [2]		*Radial Basis Function Neural Network* avec réseau de Elman pour se souvenir des évènements passés
15	[10]	NSL-KDD & KDD'99 avec tous les attributs (hybride)	/	/	99,00%		Arbre de décision (C4.5) pour mauvaise utilisation puis SVM pour anomalie
16	[16]	KDD'99 avec certains attributs sélectionnés sur base de la corrélation (CFS)	/	99,96%	99,96% [2]	FPRate = 0,01% [2], F-Measure = 0,9895 [2]	AdaBoost M1 avec MLP, KNN, DT C4,5, SVM et analyse discriminante linéaire
17	[16]	NSL-KDD avec certains attributs sélectionnés sur base de la corrélation (CFS)	99,70% [2]	99,99%	/	FPRate = 0% [2], F-Measure = 0,9984 [2]	AdaBoost M1 avec MLP, KNN, DT C4,5, SVM et analyse discriminante linéaire

TABLE 2.12 – Performances des méthodes ensemblistes

2.4.12 Résumé des performances

La table 2.13 reprend les meilleurs résultats obtenus avec chacune des méthodes. Ce tableau a pour objectif de faciliter la comparaison entre les différentes méthodes. Cette comparaison permet de conclure que les arbres de décision semblent donner de très bons résultats, tout comme les méthodes des K plus proches voisins (KNN), des réseaux bayésiens ou encore les réseaux de neurones artificiels et les méthodes ensemblistes. Les séparateurs à vaste marge, les algorithmes génétiques et les méthodes bayésiennes naïves arrivent légèrement derrière au regard des indicateurs servant de comparaison. Le peu de résultats disponibles pour les méthodes de logique floue font état de résultats peu satisfaisants.

Étant donné les résultats repris dans ce tableau, les méthodes bayésiennes naïves, de logique floue et les forêts aléatoires ne seront pas exploitées dans la suite de ce travail. Je n'implémenterai pas non plus d'algorithme génétique car cela a déjà été réalisé au sein de l'Ecole Royale Militaire par Alexandre Croix comme expliqué dans la partie 2.5 qui suit. Pour cette méthode je ne ferai donc que rapporter les résultats obtenus. Je testerai cependant les méthodes suivantes :

- Arbres de décision ;
- K plus proches voisins ;
- Séparateurs à vaste marge ;
- Réseaux de neurones artificiels ;
- Ensemble.

Article	Méthode	Dataset	Precision	Accuracy	Rappel	Autres indicateurs	Commentaires
[8]	Bayes naïf	KDD'99 avec 11 attributs sélectionnés avec CFS	99,00%	96,16%	96,20%	AUC = 0,999 ; FPRate = 0 % ; F-Measure = 0,973 ; MAE = 0,0037 ; Kappa-Stat = 0,9539	
[8]	Arbre de décision	10% KDD'99 avec tous les attributs	100,00%	99,96%	100,00%	AUC = 1 ; FPRate = 0% ; F-measure = 1 ; Kappa Statistic = 0,993 ; MAE = 0	J48
[13]	Forêt aléatoire	New Kyoto 2006+ avec partie des attributs sélectionnés selon RF	99,00%	99,00%	99,00%		
[16]	KNN	NSL-KDD	99,09%	99,17%		FPRate = 0,26 %, F-Measure = 0,9906	
[2]	SVM	CICIDS2017	/	96,80%	92,50%	FPRate = 2,3% ; F-Measure = 0,912	
[8]	ANN	KDD'99 avec 11 attributs sélectionnés avec CFS	99,30%	99,28%	99,30%	AUC = 0,999 ; FPRate = 0,1% ; F-Measure = 0,991 ; Kappa Statistic = 0,988 ; MAE = 0,001	MLP

3. La valeur a été calculée en effectuant la moyenne des résultats obtenus sur différentes attaques (smurf, ipsweep, guess password, buffer overflow)

[8]	Réseaux bayésiens	KDD'99 avec 11 attributs sélectionnés avec CFS	99,80%	99,72%	99,70%	AUC = 0,997 ; FPRate = 0% ; F-Measure = 0,997 ; Kappa Statistic = 0,9952 ; MAE = 0,0003	
[16]	Ensemble	NSL-KDD avec certains attributs sélectionnés sur base de la corrélation (CFS)	99,70%	99,99%	/	FPRate = 0 % [2], F-Measure = 0,9984 [2]	AdaBoost M1 avec MLP, KNN, DT C4,5, SVM et analyse discriminante linéaire
[7]	Algorithme génétique	Partie de KDD'99 (10,000 instances entraînement et 10,000 pour test)	/	/	93,78%	FPRate = 2,75 - 10,8	*Evolutionnary Comp - GA*
[10]	Logique floue	KDD'99 avec tous les attributs	/	/	60,55% [2]	FPRate = 2%	

TABLE 2.13 – Résumé des performances des différentes méthodes

2.5 WOWA

Comme mentionné dans l'introduction, le travail présenté à travers ce document s'inscrit dans la continuité d'un travail déjà débuté au sein de l'École Royale Militaire, lui-même constituant une partie d'un projet de plus grande envergure qui porte le nom MASFAD. MASFAD est un système multi-agent permettant la détection de menaces persistantes avancées (*Advanced Persistent Threat - APT*). Les paragraphes qui vont suivre présentent les recherches effectuées et les résultats obtenus par les chercheurs de l'École Royale Militaire. Le compte-rendu de ces recherches est disponible aux articles [24] et [25] desquels découlent en partie les explications qui vont suivre.

Les différentes recherches menées avaient pour objectif de concevoir une méthode permettant de déterminer les paramètres de la fonction WOWA. WOWA est l'abréviation de *Weighted Ordered Weighted Averaging*. Cette fonction est un opérateur d'agrégation proposé la première fois par Vicen Torra en 1996. Cet opérateur combine l'opérateur OWA (*Ordered Weighted Averaging*) avec la moyenne pondérée. Cela permet de pondérer les différentes sources d'informations tout en donnant une importance plus élevée aux sources dont le score est le plus élevé. WOWA fusionne donc un ensemble de données numériques en un seul nombre grâce à deux vecteurs de pondérations :

— w pour la moyenne pondérée ;
— p pour l'opérateur OWA.

La fonction WOWA peut donc se résumer sous la forme suivante :

$$sortie = WOWA(w, p, donnees) \tag{2.26}$$

Afin de trouver les poids idéaux, il est nécessaire de résoudre le problème d'optimisation. L'algorithme choisit pour résoudre ce problème est un algorithme génétique dont les principales étapes vont être résumées ci-après. Le fonctionnement des algorithmes génétiques a déjà été exposé dans la partie 2.4.8. Pour faire référence à ce qui a été dit dans cette partie, les gènes sont dans ce cas-ci des poids uniques et les chromosomes sont des éléments composés de deux vecteurs de poids w et p. Les vecteurs de poids contiennent plusieurs gènes dont la somme vaut 1.

2.5.1 Génération de la population initiale

La population initiale $P(0)$ est composée de N chromosomes de deux vecteurs, chacun contenant M gènes, M étant le nombre de données sources à fusionner. La méthode proposée par Croix, Debatty et Mees comporte deux méthodes d'initialisation de la population. La première est une méthode aléatoire. Elle consiste à générer aléatoirement un nombre compris entre 0 et 1 pour chaque gène. Ensuite chacun des deux vecteurs est normalisé indépendamment l'un de l'autre pour respecter les conditions précédemment énoncées. La deuxième méthode est une méthode quasi-aléatoire. La plupart des chromosomes sont générés aléatoirement tandis que pour d'autres tous les poids sont fixés à 0 sauf pour un seul où le poids est initialisé à 1. L'objectif de cette méthode est de débuter l'algorithme avec des cas particuliers afin de permettre une éventuelle meilleure convergence vers la solution.

2.5.2 Évaluation des performances

Pour déterminer le chromosome qui donne le meilleur résultat, chacun d'entre eux est évalué individuellement. Croix, Debatty et Mees ont de nouveau opté pour deux méthodes d'évaluation différentes. La première se base sur la distance. La fonction WOWA est calculée sur chaque exemple de l'ensemble d'entraînement à laquelle on soustrait la valeur disponible dans l'ensemble d'entraînement. Toutes ces différences sont ensuite additionnées et, plus la somme des distances est faible, plus le résultat est bon. La deuxième méthode se base sur l'aire sous la courbe ROC (*Area Under the ROC Curve*). Comme pour la méthode précédente, la fonction WOWA est calculée sur tous les éléments de l'ensemble d'entraînement et l'aire sous la courbe est obtenue sur base de ces résultats.

2.5.3 Sélection

Dans le but de trouver la génération $t+1$, il est nécessaire de sélectionner des éléments de la génération t. Deux méthodes ont été implémentées pour sélectionner les chromosomes parents en plus d'une méthode qualifiée d'élitiste où les deux meilleurs chromosomes sont conservés pour la génération suivante. Dans les deux premiers cas, le nombre de chromosomes parents sélectionnés dépend du paramètre de croisement (*crossover).* La première méthode appelée sélection de la roulette (*Roulette Wheel Selection - RWS*) donne une probabilité $\mathbb{P}(i)$ (équation 2.27) d'être sélectionnée, proportionnelle au score de performance $f(i)$, où i est un individu de la population.

$$\mathbb{P}(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \tag{2.27}$$

Où n est le nombre d'individus dans la population.
Cette méthode peut engendrer une convergence vers un minimum local.
L'autre méthode appelée sélection par tournoi (*Tournament Selection - TOS*) sélectionne aléatoirement deux chromosomes et conserve le meilleur d'entre eux pour la génération suivante. Pour une méthode comme pour l'autre, le processus est répété tant que le nombre de chromosomes parents n'est pas atteint.

2.5.4 Reproduction

L'étape de reproduction consiste à combiner les chromosomes précédemment sélectionnés dans le but de constituer de nouveaux individus. La première étape est de générer un nombre α pour déterminer le point de croisement.

$$\alpha = \lceil (alea * M) \rceil \tag{2.28}$$

M étant le nombre de données numériques à fusionner et *alea* un nombre aléatoire entre 0 et 1.

Ensuite, deux chromosomes qui sont destinés à la génération suivante sont sélectionnés :

$$P_{pre} = (p_{p1}, p_{p2}, ..., p_{p\alpha}, ..., p_{pM}) \tag{2.29}$$

$$P_{mre} = (p_{m1}, p_{m2}, ..., p_{m\alpha}, ..., p_{mM}) \tag{2.30}$$

Les gènes situés au point de croisement sont combinés pour créer un nouveau gène :

$$p_{nouveau1} = p_{m\alpha} - \beta[p_{m\alpha} - p_{p\alpha}] \tag{2.31}$$

$$p_{nouveau2} = p_{p\alpha} + \beta[p_{m\alpha} - p_{p\alpha}] \tag{2.32}$$

β étant un nombre aléatoire entre 0 et 1. La dernière étape est la combinaison entre les deux parents et les deux nouveaux gènes :

$$enfant_1 = (p_{m1}, p_{m2}, ..., p_{nouveau1}, ..., p_{pM}) \tag{2.33}$$

$$enfant_2 = (p_{p1}, p_{p2}, ..., p_{nouveau2}, ..., p_{mM}) \tag{2.34}$$

Ce processus est répété jusqu'à ce que la population soit constituée.

2.5.5 Mutation

La mutation est une étape importante afin d'éviter que l'algorithme ne converge trop rapidement vers un minimum local. Pour ce faire, un gène est sélectionné aléatoirement et remplacé par une valeur aléatoire entre 0 et 1 avant que le chromosome soit normalisé et ce toujours dans le but de conserver la condition qui veut que l'ensemble des gènes au sein d'un chromosome somme à un.

2.5.6 Résultats

Les différents tests menés sur cette méthode montrent que la répartition des poids dépend fortement du critère de performance utilisé. Avec les critères de distance, le module avec le poids le plus important est le module basé sur la signature tandis qu'avec le critère de l'aire sous la courbe, il correspond au module avec le poids le plus faible. Ces tests ont également

permis de montrer que l'initialisation quasi-aléatoire ne donne pas de meilleurs résultats que l'initialisation aléatoire, tout comme le paramètre du nombre d'individus dans la population qui ne semble pas avoir une grande influence sur les résultats. Le tableau 2.14 reprend les valeurs des paramètres qui permettent d'obtenir les meilleurs résultats en se basant sur le critère de l'aire sous la courbe ROC.

Paramètre	Valeur paramètre	Résultat AUC-ROC
Taille de la population	75	0,88114
Taux de croisement	40	0,88117
Taux de mutation	20	0,88125
Fonction de fitness	AUC	0,88125

TABLE 2.14 – Paramètres de l'algorithme génétique qui produisent les meilleurs résultats avec le critère ROC

L'algorithme génétique a été comparé avec les réseaux de neurones. La méthode basée sur les réseaux de neurones a également subi une étude paramétrique afin de déterminer les valeurs idéales des différents paramètres suivant :

- Fonction d'activation ;
- Nombre de neurones ;
- Taux d'apprentissage ;
- Taille du lot (*batch size*) qui correspond au nombre d'éléments qui passent à travers le réseau durant la phase d'entraînement avant que les poids ne soient modifiés ;
- Nombre d'époque (*epoch number*) qui correspond au nombre de fois que le jeu de données passera entièrement dans le réseau durant la phase d'entraînement.

Le tableau 2.15 reprend, comme pour l'algorithme génétique, les paramètres idéaux du réseau de neurones qui permettent d'obtenir les meilleurs résultats en prenant l'aire sous la courbe ROC comme évaluation.

Paramètre	Valeur paramètre	Résultat AUC-ROC
Nombre de neurones	38	0,93761
Taux d'apprentissage	0,04	0,93979
Taille du lot	2000	0,92746
Nombre d'époques	350	0,94989
Fonction d'activation	ReLu	0,94989

TABLE 2.15 – Paramètres du réseau de neurones qui produisent les meilleurs résultats avec le critère ROC

La comparaison entre la méthode se basant sur l'algorithme génétique pour déterminer les poids de *WOWA* et la méthode se basant sur les réseaux de neurones montre que cette dernière donne sans aucun doute de meilleurs résultats comme en témoigne le tableau 2.16. La moyenne des résultats est obtenue en effectuant 10 fois une validation croisée en 10 blocs autant pour l'algorithme génétique que pour la méthode des réseaux de neurones. L'étude paramétrique menée sur chacune des deux méthodes semble avoir plus de sens pour les réseaux de neurones que pour l'algorithme génétique car les différentes valeurs des paramètres ont peu d'influence sur les résultats dans ce dernier cas.

Classifieur	ROC
Algorithme génétique	0,900598
Réseau de neurones	0,946812

TABLE 2.16 – Résultats obtenus après 10 validation croisées en 10 blocs

L'article [24] mentionne, dans les travaux futurs, qu'une suite intéressante pourrait être de tester d'autres fonctions d'agrégation. Les premières recherches menées dans l'article [25] avec les réseaux de neurones s'inscrivent dans cette optique et c'est également l'objectif de ce travail. En effet, les pages qui vont suivre seront consacrées à l'implémentation et la comparaison de nouvelles méthodes d'agrégation afin d'essayer d'améliorer encore les performances de détection de ces intrusions.

Chapitre 3

Implémentation des méthodes et performances

3.1 Langage utilisé

Les pages qui vont suivre reprennent les différents résultats de certaines méthodes abordées précédemment et appliquées à la détection de *WebShell*. Les algorithmes de Machine Learning seront implémentés en Java. En effet, le projet en cours au sein de l'Ecole Royale Militaire dans lequel s'inscrit cette recherche est développé en Java. Pour des raisons d'intégration et de continuité, toutes les méthodes présentées seront donc implémentées dans ce langage.

Une librairie nommée *Smile* (*Statistical Machine Intelligence and Learning Engine*) [26] contient de nombreuses méthodes de Machine Learning implémentées en Java. Cette librairie est équivalente à la librairie *Scikit Learn* en Python. Les exemples d'utilisation de la librairie *Smile* sont beaucoup moins nombreux que pour les librairies Python équivalentes, *Smile* étant moins utilisé pour effectuer du Machine Learning. Cependant, la documentation officielle de la librairie [26] permet tout de même d'exploiter très largement les capacités de *Smile*. La librairie propose notamment les modèles suivants : KNN, SVM, MLP, arbres de décision, forêts aléatoires et les modèles de Bayes naïfs.

Une autre librairie existe également pour effectuer du Machine Learning en Java. Cette librairie est la librairie WEKA. Ce nom est ressorti à plusieurs reprises lors de mes recherches. En effet, WEKA propose aussi un logiciel qui permet de réaliser des tâches de classification directement à partir d'une interface (figure 3.1). Ce logiciel a été développé en Java par l'université de Waikato en Nouvelle-Zélande. La librairie WEKA propose notamment l'implémentation des méthodes des forêts aléatoires, des réseaux de neurones et des arbres de décision.

Un dernier outil intéressant pour effectuer du Machine Learning en Java est la librairie DL4J (*Deep Learning For Java*). Elle permet de créer un réseau de neurones couche par couche pour une personnalisation la plus complète.

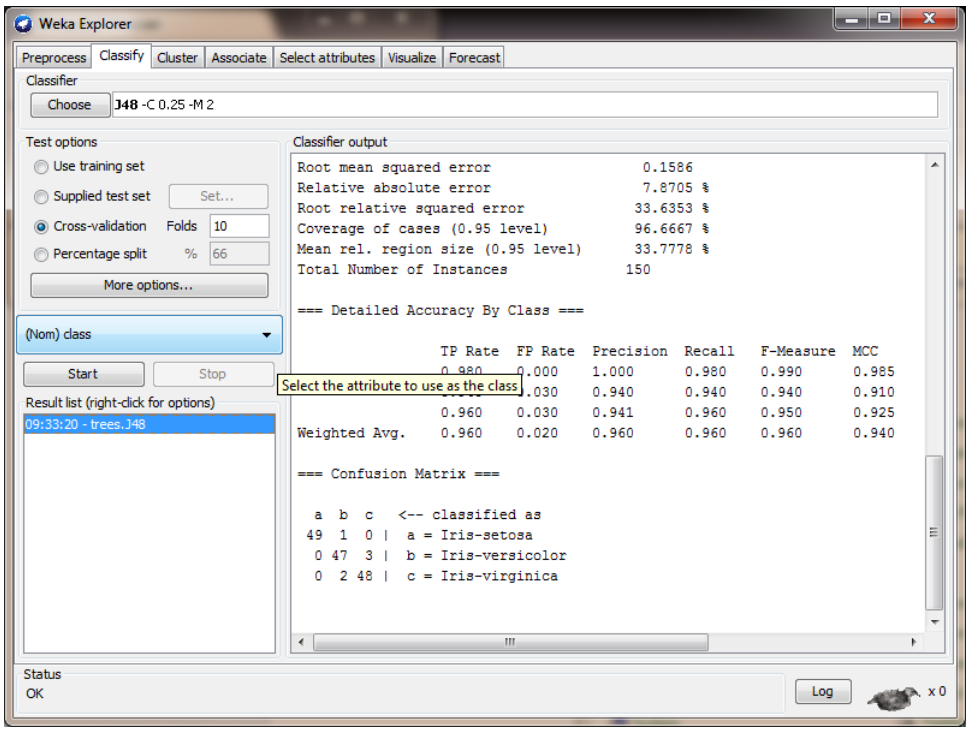

FIGURE 3.1 – Exemple d'interface du logiciel WEKA [27]

3.2 Détecteurs

Avant de commencer à développer une quelconque méthode, il est intéressant de disposer de résultats de référence à propos des 5 détecteurs décrits dans la section 2.2.7. En effet, les performances des détecteurs sont des indicateurs qui permettront, par la suite, de statuer sur l'intérêt des différentes méthodes implémentées. Des tests ont donc été effectués.

Les performances des différents détecteurs sont reprises dans les tableaux 3.1, 3.2 et 3.3 ainsi que les résultats obtenus avec des méthodes ensemblistes simples comme le vote majoritaire et la moyenne. Les tests effectués sur les différents détecteurs ont permis de déduire les forces et faiblesses de chacun d'entre eux. Les détecteurs basés sur les routines dangereuses et sur l'obscurcissement sont équivalents. Ils sont mauvais dans la détection des fichiers malveillants (Rappels de 0,266630611 et 0,333693889 respectivement) mais sont assez bons pour reconnaître les fichiers classiques (Précisions respectivement de 0,847079038 et 0,738922156). Les détecteurs basés sur les signatures et sur le hachage flou sont également équivalents. Comme les deux précédents, ils sont mauvais pour la détection des fichiers malveillants (Rappels de 0,188750676 et 0,152514873 respectivement) mais sont très bons pour reconnaître les fichiers classiques (Précisions respectivement de 0,994301994 et 0,996466431). Le dernier basé sur l'entropie est bon pour détecter les fichiers corrompus (Rappel de 0,85181179). Toutefois, il a tendance à se tromper dans la détection des fichiers sains (Précision de 0,137638731). En ce qui concerne les performances du vote majoritaire et de la moyenne, les résultats exposés dans les tableaux montrent que ces 2 méthodes ne permettent pas d'obtenir des performances réellement supérieures aux précédentes. Ces deux modèles possèdent les mêmes caractéristiques que les détecteurs basés sur les signatures et le hachage flou. Ils sont effectivement très bons pour reconnaître les fichiers sains mais sont médiocres concernant les fichiers malveillants.

Routines	0,78309987, 95% CI [0,77043553 ;0,79576421]
Signatures	0,59432897, 95% CI [0,58021737 ;0,60844057]
Entropie	0,83748274, 95% CI [0,82598144 ;0,84898404]
Hachage flou	0,59973744, 95% CI [0,58562324 ;0,61385163]
Obscurcissement	0,78536696, 95% CI [0,7727427 ;0,79799123]
Vote majoritaire	0,64755464, 95% CI [0,6335199 ;0,66158938]
Moyenne	0,62950629, 95% CI [0,61541954 ;0,64359304]

TABLE 3.1 – AUC des 5 détecteurs

Routines	0,50737308, 95% CI [0,50096851 ;0,51377523]
Signatures	0,40808899, 95% CI [0,40180911 ;0,41439902]
Entropie	0,63508845, 95% CI [0,62890034 ;0,64123224]
Hachage flou	0,27704093, 95% CI [0,27134524 ;0,28280976]
Obscurcissement	0,47493556, 95% CI [0,46854366 ;0,48133568]
Vote majoritaire	0,48564879, 95% CI [0,47924972 ;0,49205256]
Moyenne	0,46141538, 95% CI [0,45503674 ;0,46780667]

TABLE 3.2 – AUC-PR des 5 détecteurs

Routines	0,40559441, 95% CI [0,38321363 ;0,42797518]
Signatures	0,31727273, 95% CI [0,29605848 ;0,33848697]
Entropie	0,23698465, 95% CI [0,21760195 ;0,25636736]
Hachage flou	0,26454034, 95% CI [0,24443492 ;0,28464575]
Obscurcissement	0,45976155, 95% CI [0,43704477 ;0,48247833]
Vote majoritaire	0,45518966, 95% CI [0,43249068 ;0,47788865]
Moyenne	0,41133534, 95% CI [0,38890584 ;0,43376484]

TABLE 3.3 – *F-Measure* des 5 détecteurs

3.3 Séparateur à vaste marge

Sur base des différents résultats récoltés lors des recherches, la première méthode adaptée et utilisée dans le cadre de la détection de *WebShell* est la méthode des SVM. Comme annoncé, dans la section 3.1, la librairie *Smile* a été utilisée pour implémenter les SVM. Les paramètres proposés par la librairie pour cette méthode étant les suivants :

— Noyau qui peut être linéaire, gaussien, laplacien ou encore polynomial ;
— C le paramètre de pénalité de la marge qui correspond au coût de mauvaise classification. Cela signifie que plus C sera élevé, plus le SVM s'adaptera aux données d'entraînement avec pour risque de provoquer un sur-apprentissage.

En plus des paramètres de la méthode SVM, les différents noyaux possèdent des paramètres propres. Pour les noyaux gaussien et laplacien il faut ajouter le paramètre sigma qui détermine le niveau de non-linéarité du noyau (plus sigma sera élevé plus la limite de décision sera linéaire). Pour les noyaux polynomiaux, le paramètre supplémentaire est le degré du polynôme.

Finalement, en plus de ces paramètres directement liés à la méthode des SVM, les performances peuvent être influencées par deux paramètres supplémentaires dont l'*increase ratio* qui correspond au rapport d'augmentation. En effet, étant donné la faible quantité de vrais positifs dans le jeu de données, il est possible, tout comme pour la méthode WOWA d'augmenter ce

nombre en dupliquant les vrais positifs un certain nombre de fois. Le deuxième paramètre est le paramètre k, aussi appelé *fold number*, inhérent à la validation croisée. Plus ce nombre sera élevé, plus le jeu de données sera divisé en de nombreux sous-ensembles (blocs) qui serviront chacun à leur tour d'ensembles de test et d'entraînement.

Afin d'évaluer les performances de la méthode des séparateurs à vaste marge, j'ai effectué un grand nombre de tests en faisant varier un à un chaque paramètre les autres étant fixés. Cette manière de réaliser les tests nécessite de poser l'hypothèse, à priori, que tous ces paramètres ne sont pas corrélés entre eux. Durant les différents tests, les paramètres fixés prennent les valeurs suivantes :

— C = 10 ;
— Sigma = 0,1 pour les noyaux gaussien et laplacien ;
— *Degree* = 5 pour le noyau polynomial ;
— *Increase Ratio* = 5 ;
— *Fold number*(k) = 5.

Étant donné que tous les paramètres ne sont pas communs à tous les noyaux, les tests ont été séparés en 3 parties :

— Tests des noyaux laplacien et gaussien intégrant le paramètre sigma ;
— Tests du noyau polynomial intégrant la paramètre degré ;
— Tests du noyau linéaire qui n'intègre ni sigma ni degré.

3.3.1 Noyaux gaussien et laplacien

Le premier test que j'ai réalisé consiste à comparer les performances des deux noyaux, tous les autres paramètres étant identiques. Cela m'a permis, comme le montre la figure 3.2, de déduire que le noyau laplacien semble donner de meilleurs résultats (tableau 3.4). La *F-Measure* m'a également aidé à déterminer le noyau optimal, cet indicateur ayant une valeur de 0,73 pour le noyau gaussien et 0,75 pour le noyau laplacien.

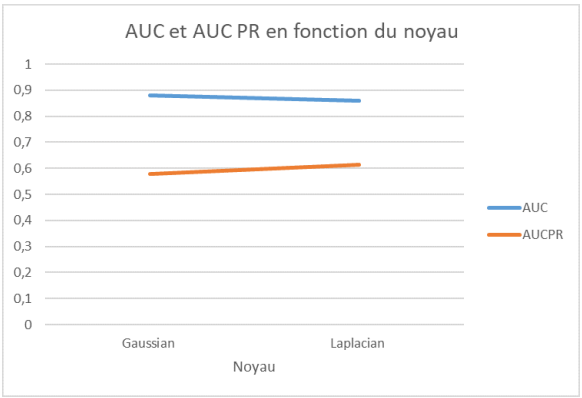

FIGURE 3.2 – Performances en fonction du noyau

J'ai ensuite effectué différents tests afin de trouver le C optimal. Le premier test (figure 3.3) m'a permis de trouver la zone dans laquelle serait située la valeur optimale qui semble être de 100 pour obtenir les meilleures performances. Le second essai (figure 3.4) a, quant à lui, permis d'affiner la recherche de cet optimum. En analysant de plus près les valeurs obtenues,

AUC	0,86003049, 95% CI [0,84915484 ;0,87090615]
AUC-PR	0,61514828, 95% CI [0,60889743 ;0,62136135]
F-Measure	0,74679749, 95% CI [0,72697661 ;0,76661838]

TABLE 3.4 – Indicateurs de performance pour le noyau laplacien

c'est encore une fois aux alentours d'une valeur de 100 pour C que la classification semble être la meilleure. Le tableau 3.5 reprend les valeurs des différents indicateurs.

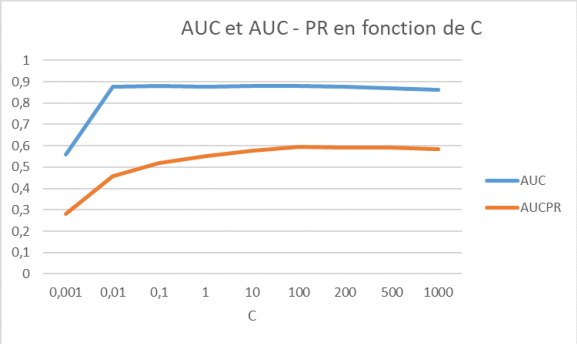

FIGURE 3.3 – Performances en fonction de C - Test 1

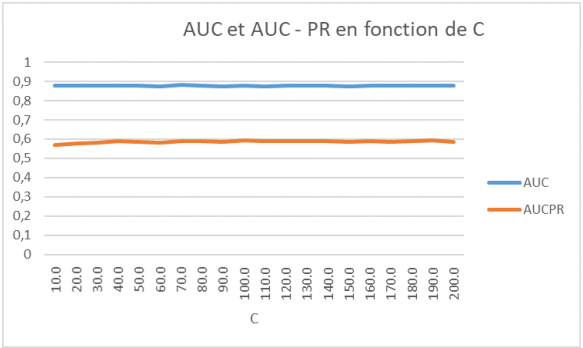

FIGURE 3.4 – Performances en fonction de C - Test 2

AUC	0,87914798, 95% CI [0,86888547 ;0,8894105]
AUC-PR	0,59422036, 95% CI [0,58791555 ;0,60049426]
F-Measure	0,74021352, 95% CI [0,72022529 ;0,76020176]

TABLE 3.5 – Indicateurs de performance pour C = 100

Tout comme pour la détermination du paramètre C optimal, la recherche de sigma a demandé plusieurs essais. Le premier essai (figure 3.5) montre assez clairement un maximum autour de sigma = 0,1. Le deuxième (figure 3.6) est, lui, moins explicite mais la valeur de 0,05 donne de meilleurs résultats. Cette valeur se trouvant à l'extrémité gauche du graphique, un nouvel essai est donc nécessaire afin de confirmer voire affiner cette valeur. Ce nouveau test (figure 3.7) permet de déterminer que les valeurs optimales de AUC, AUC-PR tout comme la *F-Measure* sont atteintes lorsque sigma vaut 0,06 (tableau 3.6).

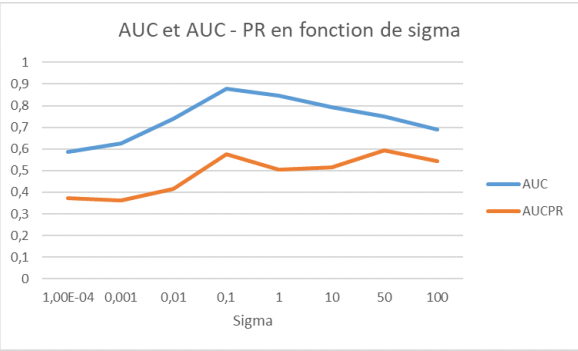

FIGURE 3.5 – Performances en fonction de sigma - Test 1

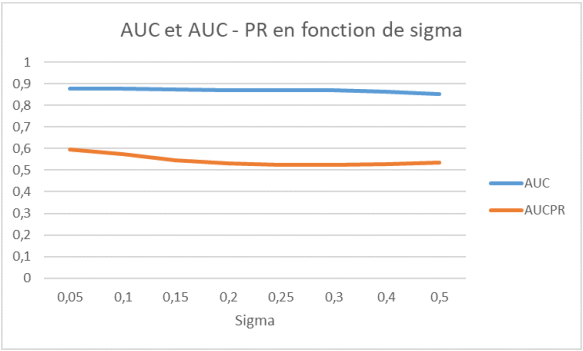

FIGURE 3.6 – Performances en fonction de sigma - Test 2

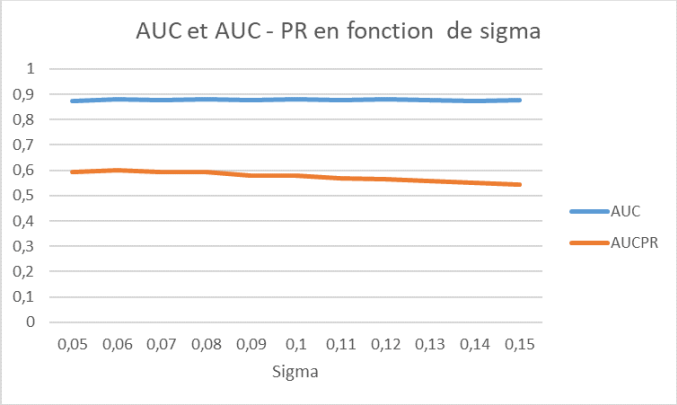

FIGURE 3.7 – Performances en fonction de sigma - Test 3

AUC	0,88023112, 95% CI [0,87000591 ;0,89045632]
AUC-PR	0,59879236, 95% CI [0,59249832 ;0,60505399]
F-Measure	0,74343949, 95% CI [0,72353251 ;0,76334647]

TABLE 3.6 – Indicateurs de performance pour sigma = 0.06

En ce qui concerne les paramètres relatifs à l'entraînement et non au noyau, les différents essais (figures 3.8 et 3.9) permettent de déduire que la valeur de *fold number* n'a aucune influence sur les résultats et que la valeur d'*increase ratio* optimale, quant à elle, vaut 1. Les tableaux 3.7 et 3.8 reprennent les valeurs des indicateurs de performance. Concernant le *fold number*, bien que sa valeur n'ait aucune influence, la valeur de 5 sera retenue. En effet, cette valeur permet d'avoir un ratio de 80% des données pour l'entraînement et de 20% pour le test, ce qui constitue un partage couramment utilisé en Machine Learning.

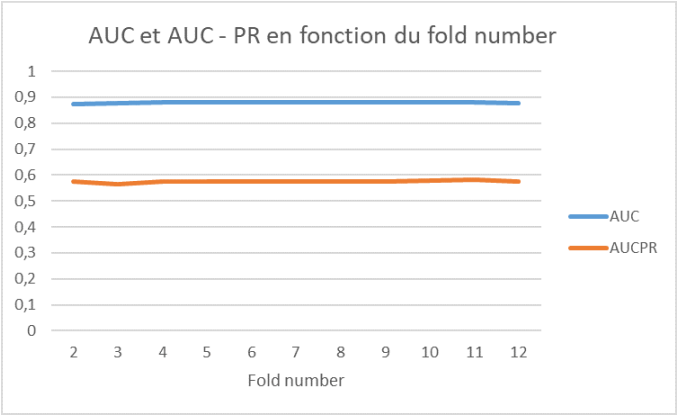

FIGURE 3.8 – Performances en fonction du *fold number*

AUC	0,88035015, 95% CI [0,87012906 ;0,89057124]
AUC-PR	0,57484563, 95% CI [0,56850143 ;0,58116527]
F-Measure	0,72781358, 95% CI [0,70752597 ;0,7481012]

TABLE 3.7 – Indicateurs de performance pour *fold number* = 5

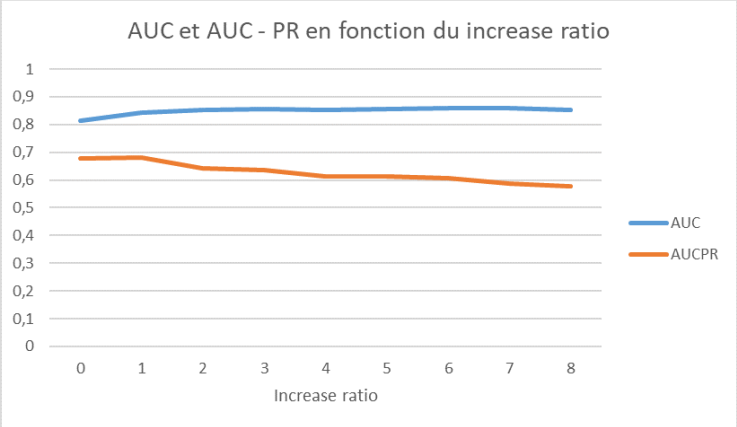

FIGURE 3.9 – Performances en fonction de *increase ratio*

AUC	0,84220453, 95% CI [0,83082635 ;0,8535827]
AUC-PR	0,67947883, 95% CI [0,67347204 ;0,68542674]
F-Measure	0,77135981, 95% CI [0,75221758 ;0,79050203]

TABLE 3.8 – Indicateurs de performance pour *increase ratio* = 1

La combinaison de ces paramètres optimaux permet d'obtenir les performances reprises dans le tableau 3.9.

AUC	0,84320762, 95% CI [0,83185611 ;0,85455912]
AUC-PR	0,6147003, 95% CI [0,60844817 ;0,6209148]
F-Measure	0,7389946, 95% CI [0,71897604 ;0,75901317]
Noyau	Laplacien
C	100
Sigma	0,06
Increase ratio	1
Fold number	5

TABLE 3.9 – Paramètres optimaux et indicateurs de performance pour SVM avec noyau laplacien

Les valeurs des différents indicateurs sont parfois inférieures lorsqu'elles sont comparées à celles des essais réalisés précédemment alors que les résultats repris dans ce tableau devraient être les meilleurs. La raison de ce phénomène réside certainement, en partie, dans l'hypothèse que nous avons posée quant à la non corrélation des paramètres. En effet, certains paramètres sont très certainement corrélés les uns avec les autres. Une manière de régler ce problème serait de tester toutes les combinaisons de tous les paramètres ce qui représenterait plusieurs dizaines de milliers de combinaisons et un temps de calcul considérable.

3.3.2 Noyau polynomial

Le paramètre central du noyau polynomial est le degré du polynôme. Pour déterminer le degré optimal, j'ai effectué un test en faisant varier sa valeur de 1 à 10 comme le montre la figure

3.10. L'indicateur AUC semble atteindre un maximum lorsque le degré vaut 3 tandis que pour l'indicateur AUC-PR , plus le degré augmente, plus sa valeur augmente (0,55 pour degré 3 à 0,62 pour degré 10). La *F-Measure* suit la même tendance que AUC-PR avec une augmentation plus forte entre le degré 1 et 3 (de 0,63 à presque 0,7). Cette tendance pousserait à prendre un degré le plus élevé possible. Le problème qui peut résulter d'un degré (trop) élevé est un temps d'apprentissage plus long et un éventuel sur-apprentissage. Le choix du degré 5 semble être un bon compromis avec des performances satisfaisantes (tableau 3.10), la dimension des données étant également de 5.

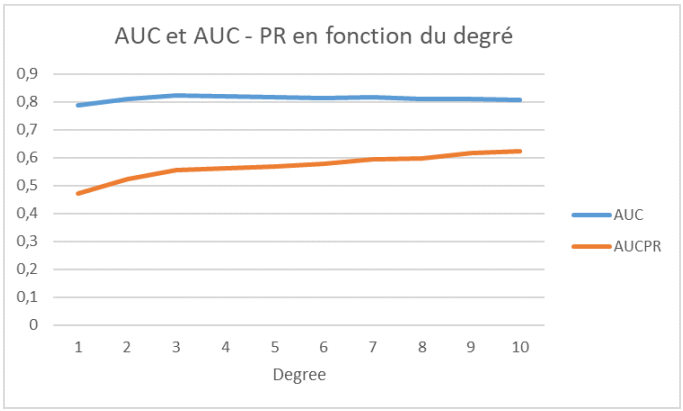

FIGURE 3.10 – Performances en fonction du degré

AUC	0,81724701, 95% CI [0,80526088 ;0,82923314]
AUC-PR	0,56838758, 95% CI [0,56203248 ;0,57472025]
F-Measure	0,69971014, 95% CI [0,67881636 ;0,72060393]

TABLE 3.10 – Indicateurs de performance pour degré = 5

Ensuite, je me suis penché sur la recherche de la valeur optimale du paramètre C. Deux essais ont été nécessaires afin de converger vers cet optimum. Le premier essai (figure 3.11) m'a permis d'observer que la valeur optimale de C se trouve aux alentours de 0,1. La deuxième série de tests (figure 3.12) ne permet pas de déduire une valeur très précise de ce paramètre bien qu'on semble distinguer un maximum très discret aux alentours de 0,15 mais rien de flagrant. La valeur de C = 0,15 sera tout de même conservée car la *F-Measure* se montre également légèrement supérieure par rapport aux valeurs de C voisines (table 3.11).

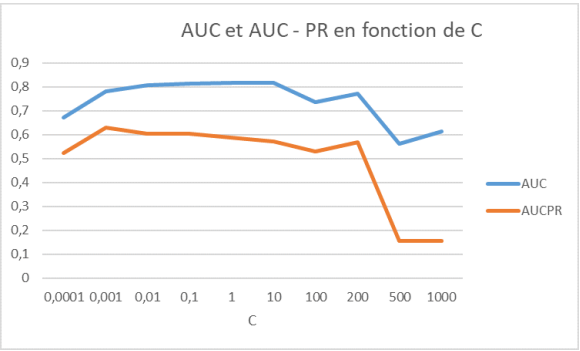

FIGURE 3.11 – Performances en fonction de C - Test 1

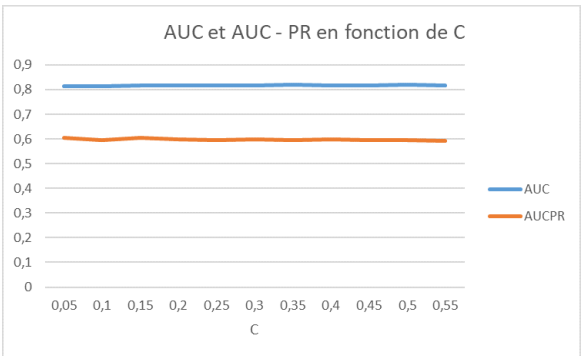

FIGURE 3.12 – Performances en fonction de C - Test 2

AUC	0,81672718, 95% CI [0,80472946 ;0,8287249]
AUC-PR	0,60340226, 95% CI [0,59711965 ;0,60965094]
F-Measure	0,71767177, 95% CI [0,69715411 ;0,73818942]

TABLE 3.11 – Indicateurs de performance pour C = 0,15

Concernant le paramètre *fold number*, la conclusion est la même que celle effectuée dans la section précédente 3.3.1. En effet, comme le montre la figure 3.13, la valeur de ce paramètre n'a aucune influence sur les performances de la méthode. La valeur retenue sera donc de 5 pour les mêmes raisons que celles évoquées dans la section 3.3.1 (résultats tableau 3.12).

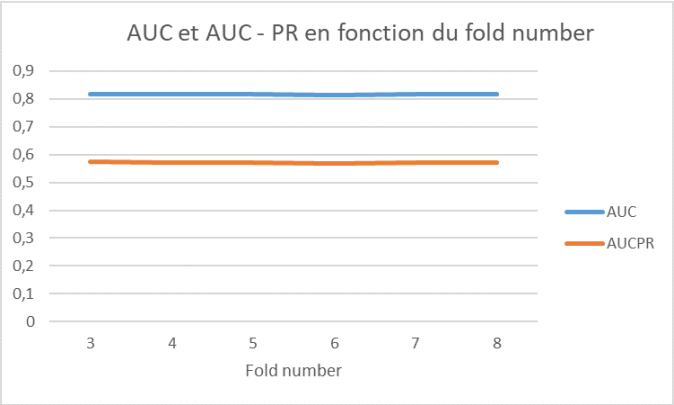

FIGURE 3.13 – Performances en fonction de *fold number*

AUC	0,81605593, 95% CI [0,80404331 ;0,82806856]
AUC-PR	0,57226401, 95% CI [0,56591532 ;0,57858899]
F-Measure	0,70110917, 95% CI [0,68024328 ;0,72197505]

TABLE 3.12 – Indicateurs de performance pour *fold number* = 5

Le paramètre *increase ratio* a, quant à lui, un impact sur les performances. Comme illustré sur la figure 3.14, les courbes AUC et AUC-PR évoluent de manières opposées. En effet, alors que la courbe AUC augmente au fur et à mesure que l'*increase ratio* augmente, la courbe AUC-PR a, quant à elle, tendance à diminuer. Dans cette situation, la *F-Measure* joue un rôle important. C'est principalement en se basant sur cet indicateur que le choix de la valeur optimale de l'*increase ratio* sera effectué. La *F-Measure* atteint un maximum lorsque le paramètre *increase ratio* vaut 2. Les indicateurs de performances relatifs à cette valeur sont repris dans le tableau 3.13.

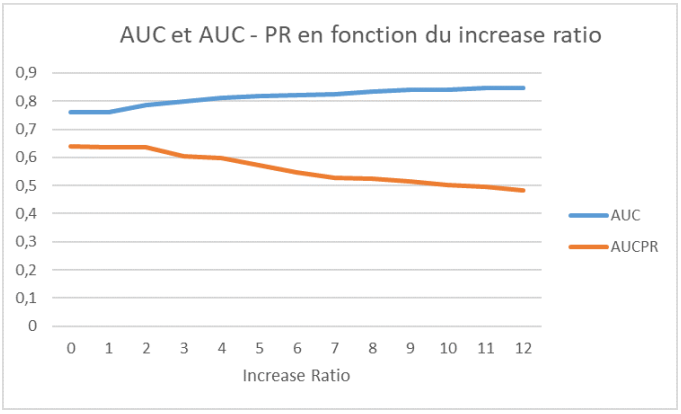

FIGURE 3.14 – Performances en fonction de *increase ratio*

AUC	0,78745425, 95% CI [0,77486748 ;0,80004102]
AUC-PR	0,63640212, 95% CI [0,63021854 ;0,64254095]
F-Measure	0,70852611, 95% CI [0,68781204 ;0,72924017]

TABLE 3.13 – Indicateurs de performance pour *increase ratio* = 2

En combinant toutes les valeurs optimales pour chacun des paramètres, cette méthode permet d'atteindre les résultats repris dans le tableau 3.14. Les conclusions à propos des résultats sont identiques à celles que nous avons pu effectuer dans la section précédente 3.3.1. Si ce n'est que, dans ce cas-ci, la valeur obtenue pour l'AUC-PR est très proche de la valeur maximale obtenue pendant les différents essais.

AUC	0,77446875, 95% CI [0,76165783 ;0,78727968]
AUC-PR	0,63499241, 95% CI [0,62880398 ;0,64113656]
F-Measure	0,69455782, 95% CI [0,67356329 ;0,71555236]
Noyau	Polynomial
C	0,15
Degré	5
Increase ratio	2
Fold number	5

TABLE 3.14 – Paramètres optimaux et indicateurs de performance pour SVM avec noyau polynomial

3.3.3 Noyau linéaire

Le seul paramètre lié au noyau linéaire à optimiser est le paramètre C. J'ai réalisé deux séries de tests afin de déterminer de manière assez précise sa valeur optimale. Le premier essai (figure 3.15) montre un maximum lorsque C = 0.01. Un deuxième test (figure 3.16) avec un panel de valeurs plus précis montre que les courbes AUC et AUC-PR évoluent de manières opposées. Plus la valeur de C augmente plus la AUC augmente et la AUC-PR diminue. La *F-Measure* joue donc un rôle d'autant plus important dans ce cas-ci car elle va permettre de déterminer le meilleur compromis. Il s'avère que le meilleur équilibre est obtenu lorsque C =

0,012 (tableau 3.15). Après cette valeur, la courbe AUC-PR semble amorcer une diminution avec une pente plus importante que précédemment.

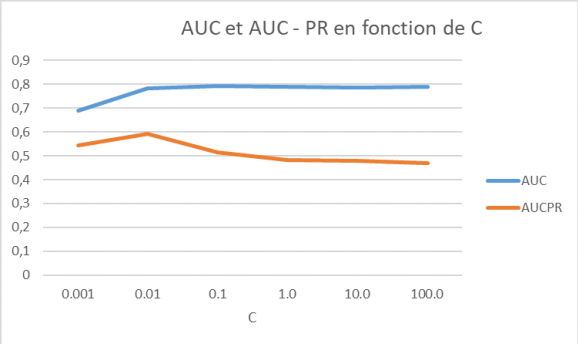

FIGURE 3.15 – Performances en fonction de C - Test 1

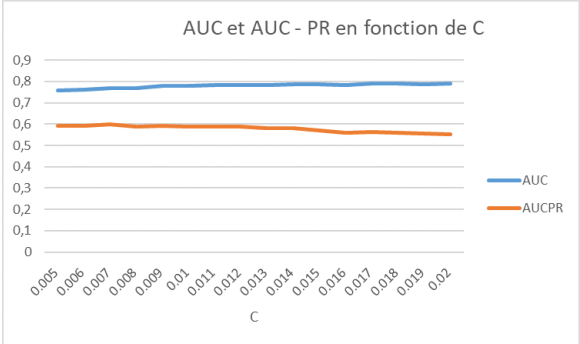

FIGURE 3.16 – Performances en fonction de C - Test 2

AUC	0,78363599, 95% CI [0,77098107 ;0,79629092]
AUC-PR	0,58720402, 95% CI [0,58088379 ;0,59349564]
F-Measure	0,68443018, 95% CI [0,66324657 ;0,70561378]

TABLE 3.15 – Indicateurs de performance pour C = 0.012

De manière identique aux conclusions des sections 3.3.1 et 3.3.2, la valeur de *fold number* n'a pas d'influence sur les performances de la méthode comme en témoigne la figure 3.17. La valeur de 5 sera donc finalement retenue comme pour les deux cas précédents. Cette valeur permet d'obtenir les résultats repris dans le tableau 3.16.

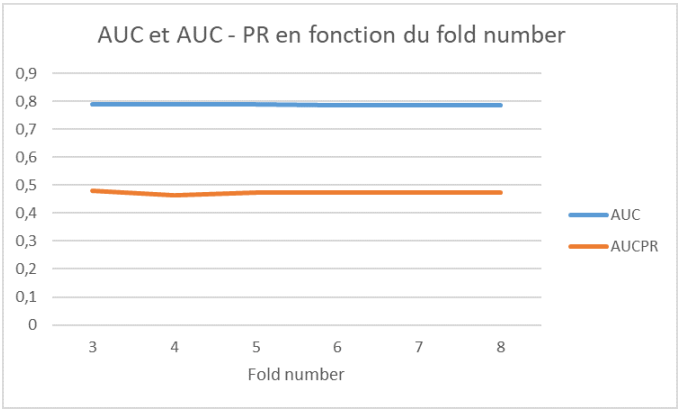

FIGURE 3.17 – Performances en fonction de *fold number*

AUC	0,78921318, 95% CI [0,77665845 ;0,80176791]
AUC-PR	0,47354408, 95% CI [0,46715333 ;0,4799435]
F-Measure	0,62956717, 95% CI [0,60755498 ;0,65157937]

TABLE 3.16 – Indicateurs de performance pour *fold number* = 5

Le dernier paramètre à tester est l'*increase ratio*. Les résultats des tests réalisés avec différentes valeurs de ce paramètre sont visibles à la figure 3.18. Les courbes AUC et AUC-PR suivent des tendances opposées, comme nous l'avions observé sur la figure 3.16, plus la valeur

du paramètre augmente plus la valeur de AUC augmente et celle de AUC-PR diminue. La *F-Measure* fera de nouveau office d'arbitre afin de définir la valeur offrant le meilleur ajustement. Ce dernier indicateur traduit qu'un *increase ratio* de 2 est le plus adapté. Les performances obtenues sont assez satisfaisantes (table 3.17).

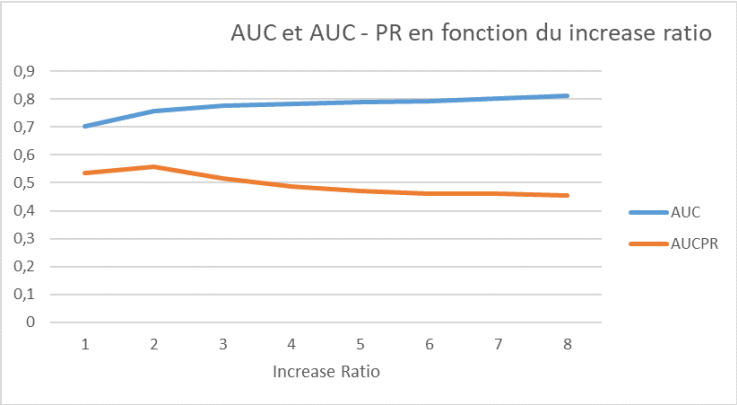

FIGURE 3.18 – Performances en fonction de *increase ratio*

AUC	0,75797451, 95% CI [0,74490889 ;0,77104014]
AUC-PR	0,55870018, 95% CI [0,55233077 ;0,56505034]
F-Measure	0,64751086, 95% CI [0,62573456 ;0,66928715]

TABLE 3.17 – Indicateurs de performance pour *increase ratio* =2

La mise en commun de tous les paramètres optimaux permet d'obtenir les résultats mentionnés dans le tableau 3.18. Encore une fois, les conclusions tirées dans les deux sections précédentes sont valables pour le noyau linéaire.

AUC	0,70057874, 95% CI [0,68686149 ;0,714296]
AUC-PR	0,54141566, 95% CI [0,53502681 ;0,54779092]
F-Measure	0,56610942, 95% CI [0,54351881 ;0,58870003]
Noyau	Linéaire
C	0,012
Increase ratio	2
Fold number	5

TABLE 3.18 – Paramètres optimaux et indicateurs de performance pour SVM avec noyau linéaire

En plus de la méthode adoptée précédemment pour déterminer la valeur optimale de chaque paramètre, l'approche par recherche aléatoire semble intéressante à explorer. Cette approche consiste à définir un ensemble de valeurs possibles pour chaque paramètre et laisser le hasard choisir une valeur pour chacun d'entre eux afin de former une nouvelle combinaison de paramètres très probablement inédite. Après avoir réalisé 20 itérations en suivant ce processus, le meilleur résultat est conservé. Ce dernier permet d'obtenir les performances reprises dans le tableau 3.19 ainsi que la combinaison de paramètres qui a permis d'y parvenir.

AUC	0,85467498, 95% CI [0,84364181 ;0,86570815]
AUC-PR	0,65348983, 95% CI [0,64736973 ;0,65955956]
F-Measure	0,76566524, 95% CI [0,74635776 ;0,78497271]
Noyau	Laplacien
C	100
Sigma	1
Increase ratio	2
Fold number	8

TABLE 3.19 – Indicateurs de performance pour SVM avec paramètres obtenus via la recherche aléatoire

3.3.4 Performances générales des séparateurs à vaste marge

Les séparateurs à vaste marge permettent d'obtenir des résultats satisfaisants dans l'ensemble. Notons que, en ne considérant que la méthode de recherche des paramètres optimaux, les meilleurs résultats sont obtenus avec le noyau laplacien. Cette méthode de classification pourrait donc être utilisée avec les paramètres indiqués dans la table 3.9. Ces paramètres ont été définis en tant que paramètres optimaux mais, comme déjà dit plus haut, l'hypothèse initiale qui pose la non-corrélation de ces paramètres semble ne pas être tout à fait correcte. La méthode utilisée avec ses paramètres réellement optimaux pourrait donner des résultats de meilleure qualité que ceux obtenus dans cette recherche. Les tests effectués avec la méthode aléatoire de recherche des paramètres donnent des résultats équivalents à ceux repris dans le tableau 3.9 voire légèrement supérieurs. Cependant, les intervalles de confiance à 95% ne permettent pas de conclure si une méthode est réellement meilleure qu'une autre.

3.4 K plus proches voisins

La deuxième méthode que j'ai décidé d'adapter pour effectuer la détection des *WebShell* est la méthode des K plus proches voisins. Cette méthode a montré de très bons résultats dans les articles précédemment analysés. De manière similaire à la méthode des séparateurs à vaste marge, j'ai utilisé la librairie *Smile* qui propose également la méthode des K plus proches voisins. Les paramètres relatifs à cette méthode sont au nombre de 2 :

- K qui définit le nombre de voisins à prendre en compte pour la classification ;
- Méthode de calcul de la distance. En effet pour choisir les K plus proches voisins, différentes méthodes de calcul de distance peuvent être utilisées (Euclidienne, Chebyshev, Jensen-Shannon, Manhattan, Minkowski et corrélation).

La distance de Minkowski prend en compte un paramètre supplémentaire p qui définit l'ordre de la distance de Minkowski. La distance de Minkowski peut s'exprimer de la manière suivante :

$$d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p} \tag{3.1}$$

Où $x = (x_1, x_2, ..., x_n)$ et $y = (y_1, y_2, ..., y_n)$ sont deux points.

La distance euclidenne peut se calculer avec la formule 3.1 avec le paramètre $p = 2$ tandis que pour la distance de Manhattan $p = 1$.

La distance de Chebyshev se calcule de la manière suivante :

$$d(x, y) = \max_i(|x_i - y_i|) \tag{3.2}$$

Elle peut aussi s'exprimer en calculant la limite de $p \to \infty$ de la formule 3.1.

Lorsque la corrélation est utilisée avec la librairie *Smile*, la méthode associée est la corrélation de Pearson qui peut se calculer de la manière suivante [28] :

$$Correlation(X, Y) = \frac{s_{xy}}{s_x * s_y} \tag{3.3}$$

Où $s_{x,y}$ correspond à la covariance entre X et Y :

$$s_{x,y} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}) \tag{3.4}$$

Et s_x et s_y sont les écarts types de X et Y, respectivement.

Comme expliqué dans la méthode précédente 3.3, deux paramètres supplémentaires sont pris en compte qui sont *increase ratio* et *fold number*.

L'hypothèse précédemment évoquée (section 3.3) de non-corrélation entre les différents paramètres est toujours d'application. Cette dernière nous permet, une nouvelle fois, de faire varier les paramètres les uns après les autres, tous les autres étant fixes. Les valeurs par défaut des paramètres sont les suivantes :

- Distance : Euclidienne ;
- K = 5 ;
- p = 3 pour la distance de Minkowski ;
- *Increase ratio* = 5 ;
- *Fold number* = 5.

Le premier paramètre testé est la méthode de calcul de distance. Comme cité précédemment, les différentes méthodes testées sont les distances Euclidienne, de Chebyshev, de Jensen-Shannon, de Manhattan et de Minkowski ainsi que la distance basée sur la corrélation. La figure 3.19 montre que la méthode de calcul de la distance n'a pas d'influence sur l'AUC mais que l'AUC-PR est légèrement meilleur avec la méthode de la corrélation. Cette dernière est donc celle qui donne le meilleur résultat (table 3.20). La raison d'un meilleur résultat fourni avec la corrélation se trouve probablement dans le fait que le calcul se base sur la covariance entre les différentes caractéristiques des données. Tandis que pour les autres mesures de distance, seule la correspondance paire à paire entre les différentes caractéristiques sont prises en compte.

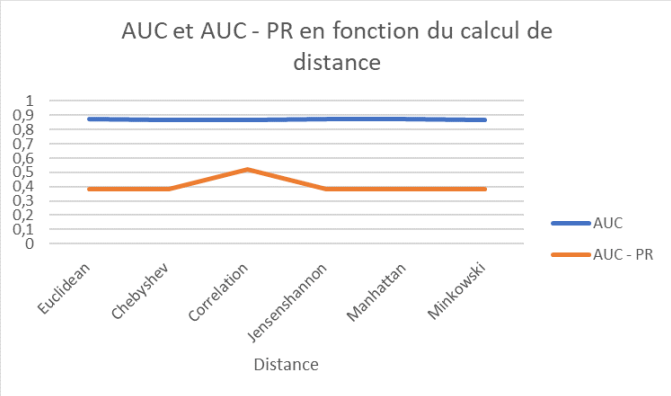

FIGURE 3.19 – Performances en fonction de la méthode de calcul de la distance

AUC	0,86963915, 95% CI [0,85906129 ;0,88021701]
AUC-PR	0,51815939, 95% CI [0,51175658 ;0,52455624]
F-Measure	0,5590187, 95% CI [0,53638733 ;0,58165007]

TABLE 3.20 – Indicateurs de performance pour la corrélation

Lorsque la méthode de calcul de distance choisie est la distance de Minkowski, le paramètre p vient s'ajouter à l'équation. La figure 3.20 indique que ce paramètre n'a aucune influence sur les performances de la méthode des K plus proches voisins et ne permet pas de dépasser les valeurs obtenues avec la méthode de calcul basée sur la corrélation. A titre indicatif, la table 3.21 reprend les résultats lorsque $p = 3$.

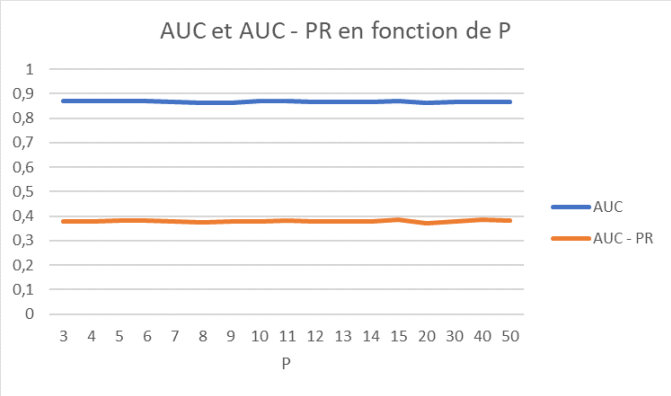

FIGURE 3.20 – Performances en fonction de p

AUC	0,86864159, 95% CI [0,85803186 ;0,87925131]
AUC-PR	0,37957894, 95% CI [0,3733831 ;0,38581429]
F-Measure	0,56379279, 95% CI [0,54118835 ;0,58639723]

TABLE 3.21 – Indicateurs de performance pour p = 3

Ensuite, j'ai testé un large éventail de valeurs pour le paramètre K comme le montre la figure 3.21. L'augmentation de la valeur du paramètre K entraîne une augmentation des indicateurs AUC et AUC-PR avec une stabilisation de AUC aux alentours de 0,925. La stabilisation des deux indicateurs a lieu à partir de la valeur de K = 50. Après cette valeur, les fluctuations et

les gains sont très faibles. D'un point de vue du temps de calcul, il est préférable d'opter pour une valeur de K la plus petite possible car plus cette valeur augmente plus le temps de calcul augmente également. La valeur de K retenue est donc de 50. Les valeurs des indicateurs de performances relatives à cette valeur de K sont disponibles dans la table 3.22.

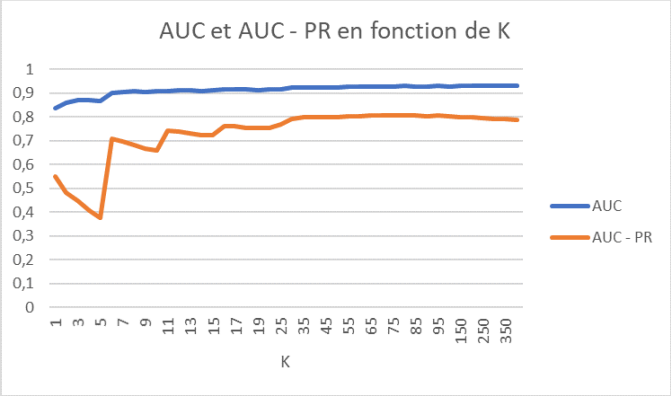

FIGURE 3.21 – Performances en fonction de K

AUC	0,9247346, 95% CI [0,9163473 ;0,93312189]
AUC-PR	0,80040513, 95% CI [0,79523618 ;0,80547552]
F-Measure	0,7048822, 95% CI [0,68409272 ;0,72567167]

TABLE 3.22 – Indicateurs de performance pour k = 50

Concernant les paramètres relatifs à l'apprentissage, comme c'était le cas pour les différents tests des séparateurs à vaste marge (section 3.3), le paramètre *fold number* n'a aucune influence sur les résultats (voir figure 3.22). Comme précédemment, la valeur de 5 sera conservée pour le *fold number* (table 3.23). Pour l'*increase ratio*, la figure 3.23, montre une certaine stabilité de l'indicateur AUC avec une légère diminution entre les valeurs 4 et 5 tandis que l'indicateur AUC-PR chute fortement aux alentours de ces valeurs alors qu'il diminuait déjà avec des valeurs précédentes. La valeur à laquelle a lieu la chute est à mettre en relation avec la valeur de K. En effet, la chute a lieu entre les valeurs 4 et 5 et la valeur de K choisie pour le test est de 5. La figure 3.24 confirme la corrélation entre les deux paramètres. Pour rappel l'*increase ratio* augmente artificiellement le nombre d'instances positives dans l'ensemble d'entraînement en dupliquant un certain nombre de fois les éléments positifs déjà présents dans cet ensemble. En analysant les données de plus près, on se rend rapidement compte que le nombre de faux positifs augmente fortement à partir d'un *increase ratio* valant $\lfloor K/2 \rfloor + 1$. Cela s'explique facilement par le fait que lorsqu'une instance, réellement négative, de l'ensemble de test se trouve proche d'une instance du modèle qui est positive, elle se trouve automatiquement à proximité de plusieurs instances positives car elles ont été dupliquées. L'instance normalement négative est labellisée comme étant positive car le nombre de positifs à proximité est plus important que le nombre de négatif d'où l'augmentation importante des faux positifs et la diminution de la précision. La précision chutant également, cela a évidemment un impact sur la courbe AUC-PR mais avec un retard. La chute n'apparaît donc sur la courbe qu'à partir d'un *increase ratio* équivalent à K.

Pour cette méthode, le choix est donc de fixer ce paramètre à 0 et de ne pas effectuer d'augmentation artificielle des vrais positifs (table 3.24).

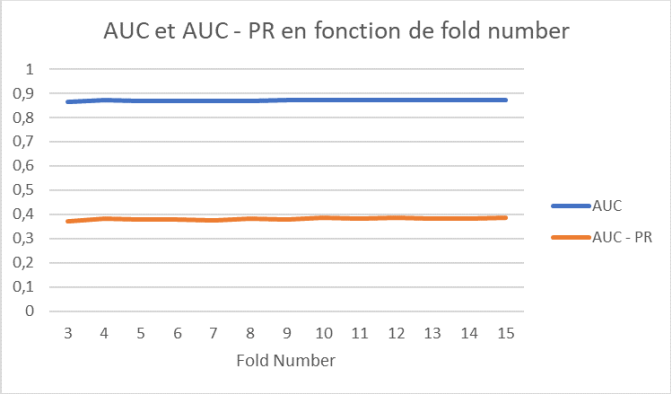

FIGURE 3.22 – Performances en fonction du *fold number*

AUC	0,86968193, 95% CI [0,85910544 ;0,88025842]
AUC-PR	0,37807805, 95% CI [0,37188726 ;0,38430885]
F-Measure	0,56121894, 95% CI [0,53859971 ;0,58383816]

TABLE 3.23 – Indicateurs de performance pour *fold number* = 5

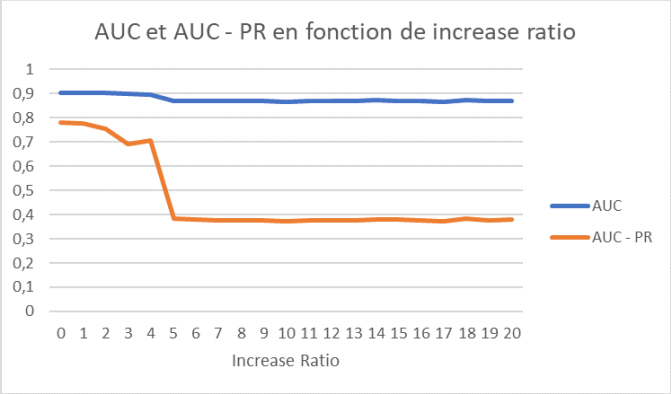

FIGURE 3.23 – Performances en fonction de *increase ratio*

AUC	0,90442561, 95% CI [0,89511772 ;0,91373351]
AUC-PR	0,78106182, 95% CI [0,77571895 ;0,78631248]
F-Measure	0,75438596, 95% CI [0,73476542 ;0,77400651]

TABLE 3.24 – Indicateurs de performance pour *increase ratio* = 0

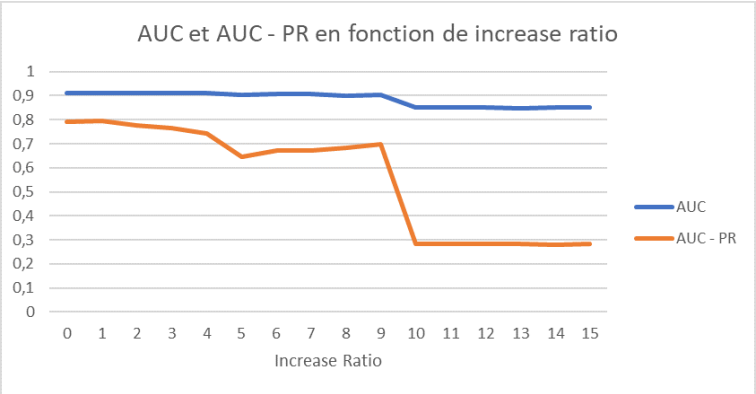

FIGURE 3.24 – Performances en fonction du *increase ratio* avec K = 10

La combinaison de tous les paramètres optimaux, auxquels les différents essais ont permis d'aboutir, ne permettent pas d'atteindre des performances supérieures à celles obtenues durant ces tests (tableau 3.25). Une des raisons de ce phénomène réside, encore une fois, en partie dans le fait que l'hypothèse posée à priori que tous les paramètres sont indépendants les uns des autres n'est probablement pas exacte.

AUC	0,91453308, 95% CI [0,90290391 ;0,92616225]
AUC-PR	0,73432813, 95% CI [0,72863229 ;0,73994708]
F-Measure	0,6408499, 95% CI [0,61209026 ;0,66960953]
Distance	Correlation
K	50
Fold number	5
Increase Ratio	0

TABLE 3.25 – Paramètres optimaux et indicateurs de performance pour KNN

Les résultats obtenus avec la recherche aléatoire des paramètres, après avoir effectué 20 itérations différentes, sont repris dans le tableau 3.26. Mis à part pour l'indicateur AUC pour lequel les intervalles de confiance à 95% se recouvrent légèrement, les autres indicateurs sont en faveur des résultats de la recherche aléatoire, et cela au détriment de la recherche des paramètres optimaux via la méthode exposée dans les paragraphes précédents.

AUC	0,93107633, 95% CI [0,92301248 ;0,93914018]
AUC-PR	0,79464864, 95% CI [0,78942606 ;0,79977455]
F-Measure	0,7267184, 95% CI [0,70640532 ;0,74703149]
Distance	Jensen-Shannon
K	90
Fold number	5
Increase Ratio	2

TABLE 3.26 – Indicateurs de performance pour KNN avec paramètres obtenus via la recherche aléatoire

En comparaison avec la méthode des SVM, les indicateurs AUC et AUC-PR sont meilleurs lorsque la méthode employée est celle des K plus proches voisins tandis que la *F-Measure* est supérieure pour la méthode SVM (tableau 3.9).

3.5 Arbres de décision

La troisième méthode implémentée lors de ce travail est celle des arbres de décision. Contrairement aux deux méthodes précédentes, je n'ai pas utilisé la librairie *Smile* pour implémenter les arbres de décision bien que cette dernière en offre la possibilité. En effet, la documentation sur cette méthode et sur les paramètres impliqués ne me permettaient pas de maîtriser l'implémentation comme je le désirais pour réaliser un code propre et exploitable par la suite. Je me suis donc dirigé vers la librairie WEKA [29]. WEKA propose un ensemble d'outils de visualisation, d'analyse de données et de modélisation prédictive sous forme d'une interface graphique ou d'une API Java. Cet outil a été cité et utilisé à plusieurs reprises dans les articles que j'ai

parcouru en amont de ce travail. Cette librairie propose l'implémentation de l'algorithme C4.5 cité dans la section 2.4.2. Les paramètres utilisés sont les suivants :

— U qui définit si les arbres seront élagués ou pas ;
— O qui définit si l'arbre sera réduit ou pas ;
— C qui définit le seuil de confiance de l'élagage ;
— M qui définit le nombre minimum d'instances par feuille ;
— B qui définit si les séparations sont exclusivement binaires ou pas.

A ces paramètres propres aux arbres de décision, le paramètre *fold number* peut également être ajouté.

Afin d'effectuer les différents essais sur les paramètres pour identifier les valeurs optimales de ceux-ci, l'hypothèse de non-corrélation entre paramètres est toujours d'application. Les valeurs des paramètres utilisées par défaut sont :

— Arbres élagués avec un taux de confiance (C) de 0,25 ;
— Arbres réduits ;
— M = 2 ;
— Séparations pas exclusivement binaires ;
— *Fold number* = 5.

Le premier paramètre sur lequel j'ai décidé de m'attarder est celui qui définit si, oui ou non, les arbres générés seront réduits. Étant donné les tendances contraires entre les 2 courbes observées sur la figure 3.25, le choix de la meilleure option se révèle un peu plus complexe. La tendance croissante de la courbe AUC-PR semble toutefois prendre le dessus sur la diminution de la courbe AUC. La *F-Measure* permet de confirmer que les arbres non réduits obtiennent de meilleures performances, visibles dans le tableau 3.27.

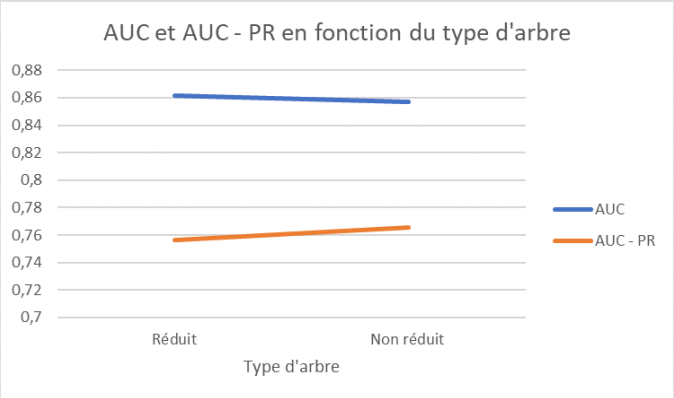

FIGURE 3.25 – Performances en fonction du type d'arbre

AUC	0,8569129, 95% CI [0,84594484 ;0,86788096]
AUC-PR	0,76539483, 95% CI [0,7599236 ;0,77077899]
F-Measure	0,76016388, 95% CI [0,74070139 ;0,77962638]

TABLE 3.27 – Indicateurs de performance pour les arbres non réduits

Contrairement au paramètre concernant la réduction des arbres, celui relatif à l'élagage ne laisse pas place à la discussion. En effet, les deux indicateurs montrent de meilleurs résultats (table 3.28) lorsque les arbres ne sont pas élagués comme le montre la figure 3.26.

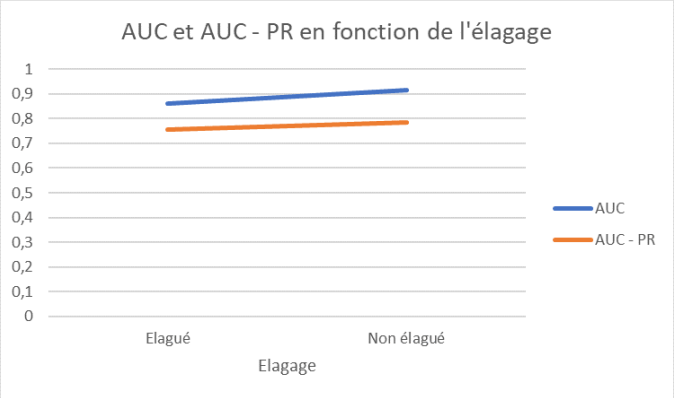

FIGURE 3.26 – Performances en fonction de l'élagage

AUC	0,91499339, 95% CI [0,90614449 ;0,92384228]
AUC-PR	0,7835358, 95% CI [0,77821419 ;0,78876437]
F-Measure	0,76806315, 95% CI [0,74882466 ;0,78730163]

TABLE 3.28 – Indicateurs de performance pour les arbres non élagués

Ensuite, comme le montre la figure 3.27, le nombre minimal d'instances par feuille (*minimum number of instances*) ne semble pas avoir une influence considérable sur les performances bien qu'un léger maximum puisse être observé lorsque ce paramètre prend une valeur de 6. En plus d'avoir une influence sur le nombre d'instances par feuille, ce paramètre a, logiquement, une grande influence sur la taille de l'arbre. Effectivement, plus sa valeur sera élevée plus la profondeur de l'arbre sera réduite. Par exemple, lorsque le paramètre est fixé à 2, la profondeur des arbres varie entre 130 et 150 tandis que, pour une valeur de 30, la profondeur est réduite à des valeurs qui oscillent entre 20 et 40. La valeur optimale de ce paramètre, que nous retiendrons dans ce cas-ci, reste tout de même 6.

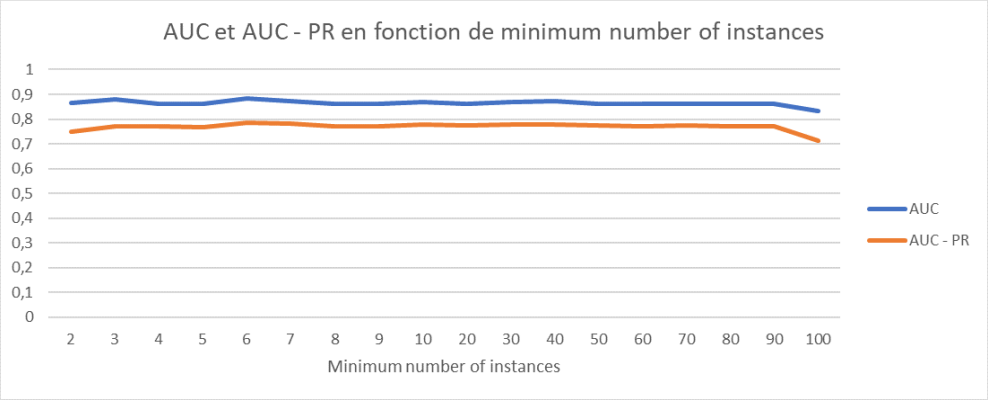

FIGURE 3.27 – Performances en fonction de *minimum number of instances*

AUC	0,88269031, 95% CI [0,87255091 ;0,8928297]
AUC-PR	0,78693314, 95% CI [0,78164119 ;0,79213094]
F-Measure	0,76102374, 95% CI [0,74158518 ;0,7804623]

TABLE 3.29 – Indicateurs de performance pour *minimum number of instances* = 6

Comme exposé précédemment, les arbres de décision semblent donner de meilleurs résultats lorsque les arbres ne sont pas élagués. Cependant, modifier la valeur du taux de confiance de l'élagage pourrait aboutir à trouver de meilleurs résultats avec élagage plutôt que sans. La figure 3.28 montre que lorsque ce taux de confiance dépasse 0,5 la courbe AUC stagne tandis que la courbe AUC-PR montre un léger mieux lorsque sa valeur est de 0,6. Cette dernière valeur sera donc retenue comme valeur optimale de *pruning confidence* (C). En comparant les valeurs des indicateurs de performances du tableau 3.30 (arbres élagués) avec ceux du tableau 3.28 (arbres non élagués), la différence de résultats est minime mais les arbres non élagués donnent des résultats globalement légèrement meilleurs.

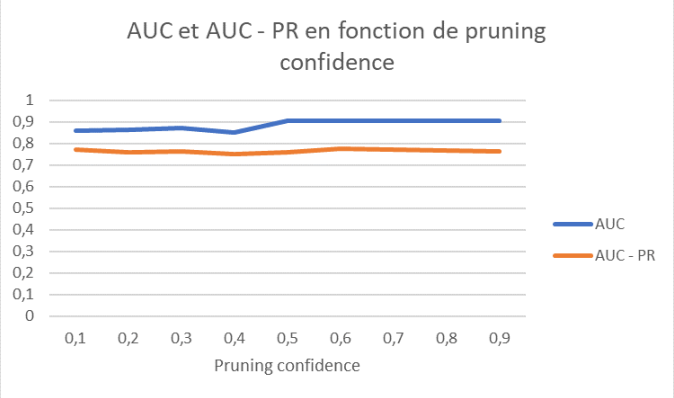

FIGURE 3.28 – Performances en fonction de pruning confidence

AUC	0,90542995, 95% CI [0,89616398 ;0,91469592]
AUC-PR	0,77679429, 95% CI [0,77141539 ;0,78208238]
F-Measure	0,7690898, 95% CI [0,74988111 ;0,78829849]

TABLE 3.30 – Indicateurs de performance pour pruning confidence = 0.6

Concernant le type de séparation, la figure 3.29 permet d'effectuer la comparaison entre les arbres où les séparations aux feuilles sont exclusivement binaires et celles qui ne le sont pas. L'indicateur AUC-PR ne montre pas de grandes différences entre les deux possibilités tandis que la courbe AUC montre une meilleure performance lorsque les séparations sont exclusivement binaires. Les performances relatives à cette situation optimale sont visibles dans le tableau 3.31.

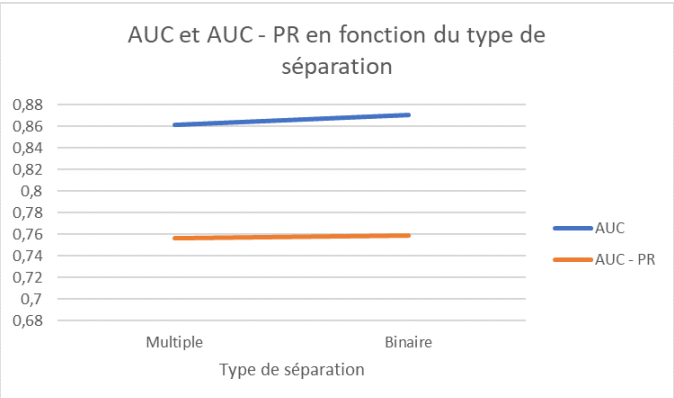

FIGURE 3.29 – Performances en fonction du type de séparation

AUC	0,87058771, 95% CI [0,86004035 ;0,88113507]
AUC-PR	0,75851636, 95% CI [0,75299208 ;0,76395582]
F-Measure	0,76228501, 95% CI [0,74288176 ;0,78168827]

TABLE 3.31 – Indicateurs de performance avec séparations exclusivement binaires

Enfin, pour la méthode des arbres de décision, le seul paramètre testé, extérieur aux arbres eux-mêmes, est le *fold number*. Contrairement aux observations qui ont pu être effectuées pour les méthodes précédentes (section 3.3 et 3.4), ce paramètre a une influence sur les performances des arbres de décision. En effet, que ce soit pour l'indicateur AUC ou AUC-PR les valeurs les plus élevées, disponibles dans le tableau 3.32, sont obtenues avec un *fold number* valant 8.

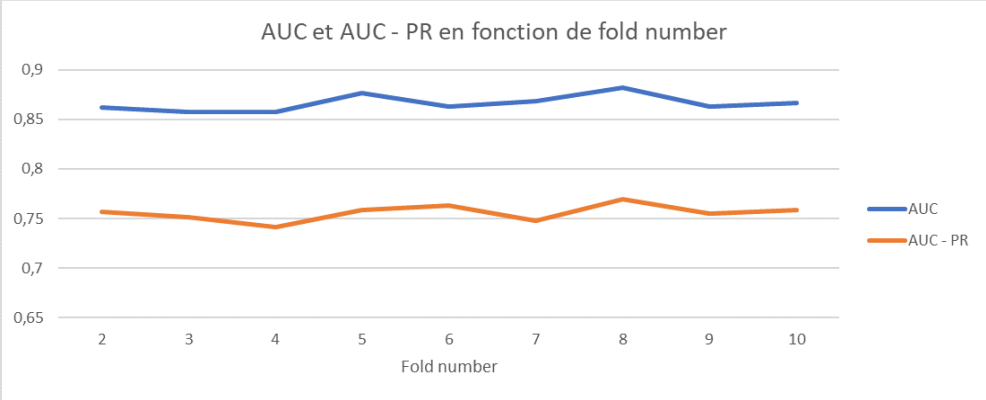

FIGURE 3.30 – Performances en fonction de *fold number*

AUC	0,88239819, 95% CI [0,87224851 ;0,89254786]
AUC-PR	0,76971949, 95% CI [0,76428263 ;0,77506786]
F-Measure	0,76448829, 95% CI [0,74514727 ;0,7838293]

TABLE 3.32 – Indicateurs de performance avec *fold number* = 8

En appliquant la méthode des arbres de décision avec tous les paramètres optimaux déterminés ci-dessus, les performances réalisées sont celles visibles dans le tableau 3.33. La combinaison de tous les paramètres optimaux permet d'obtenir des performances supérieures à toutes celles obtenues précédemment.

AUC	0,91181024, 95% CI [0,90170452 ;0,92191595]
AUC-PR	0,80257995, 95% CI [0,79743169 ;0,80762893]
F-Measure	0,76723096, 95% CI [0,74554622 ;0,78891569]
Elagage	Non
Réduction	Non
Séparation	Exclusivement binaire
Min # instances	6
Fold number	8

TABLE 3.33 – Paramètres optimaux et indicateurs de performance pour les arbres de décision

L'approche aléatoire, utilisée pour établir de nouvelles combinaisons de paramètres, donne des résultats (tableau 3.34) comparables, en tous points, avec ceux exposés dans le tableau 3.33.

AUC	0,91084653, 95% CI [0,9018127 ;0,91988036]
AUC-PR	0,80681082, 95% CI [0,8017035 ;0,81181746]
F-Measure	0,76070219, 95% CI [0,74125466 ;0,78014971]
Elagage	Non
Réduction	Oui
Séparation	Multiple
Min # instances	50
Fold number	7

TABLE 3.34 – Indicateurs de performance pour les arbres de décision avec paramètres obtenus via la recherche aléatoire

Les arbres de décision permettent d'obtenir des performances encore plus satisfaisantes que les deux méthodes implémentées précédemment (SVM 3.9 et KNN 3.25) que ce soit au niveau de l'AUC, l'AUC-PR ou encore de la *F-Measure*.

3.6 Perceptron multicouche

Le perceptron multicouche est la quatrième méthode implémentée dans ce travail. Pour implémenter les réseaux de neurones, Java propose la suite Deeplearning4j (DL4J). Ce framework permet de former et entraîner des réseaux de neurones en Java tout en interagissant avec l'environnement Python [30]. Dans son article [25] de comparaison entre la méthode WOWA et les réseaux de neurones, Alexandre Croix avait déjà utilisé cette librairie pour implémenter la méthode sur les réseaux de neurones. Je suis donc parti de ce qu'il avait réalisé pour implémenter la méthode du perceptron multicouche. Les paramètres relatifs à cette méthode sont les suivants :

- Fonction d'activation qui gère la valeur de sortie du neurone. La valeur d'entrée du neurone est modifiée en fonction des paramètres de ce neurone. Cette valeur sert ensuite d'entrée à la fonction d'activation. Les différentes fonctions utilisées sont la sigmoïde, ReLU et tangente hyperbolique (Tanh) ;
- Algorithme d'optimisation est la méthode d'optimisation utilisée afin de mettre à jour les poids du réseau de neurones. Les méthodes utilisées dans ce travail sont la descente du gradient linéaire, le gradient conjugué et la descente du gradient stochastique ;
- Taux d'apprentissage qui contrôle la manière dont le modèle est adapté en fonction de l'erreur calculée à chaque mise à jour de ce modèle. Une valeur trop élevée peut mener à un modèle sous-optimal tandis qu'une valeur trop faible va considérablement augmenter le temps d'entraînement et provoquer un phénomène de sur-apprentissage ;
- Nombres de neurones qui correspond au nombre de neurones dans la couche cachée. Un nombre de neurones trop élevé peut aussi engranger une situation de sur-apprentissage ;
- Taille du batch qui définit le nombre d'instances des données d'entraînement qui sont traitées par le modèle avant que les poids du modèle ne soient mis à jour ;
- Nombre d'epochs correspondant au nombre de fois où toutes les instances de l'ensemble d'entraînement vont être traitées par le modèle durant la phase d'entraînement.

A ces paramètres, les deux paramètres *fold number* et *increase ratio* sont ajoutés.

Dans son article [25], Alexandre Croix mentionne les valeurs optimales pour certains para-

mètres déterminés sur le même jeu de données que celui utilisé dans ce travail (tableau 3.35). Parmi ces paramètres, je conserverai les valeurs optimales obtenues pour le nombre de neurones, la fonction d'activation et le taux d'apprentissage. En effet, les tests ont été effectués avec Google Colab et, comme Alexandre Croix le mentionne dans son article, cet outil ne permet pas de réaliser des tests avec un nombre d'epochs supérieur à 350 et une taille de batch supérieure à 2000 en raison du temps d'apprentissage trop long nécessaire pour entraîner un réseau de neurones avec de tels paramètres. Je testerai donc des valeurs supérieures pour ces deux paramètres. Afin d'effectuer les tests nécessaires pour déterminer les valeurs optimales, je pose l'hypothèse à priori, comme pour les 3 méthodes précédentes, que les paramètres ne sont pas corrélés entre eux, ce qui me permet de tester différentes valeurs d'un paramètre tous les autres étant fixés. Les valeurs fixées aux différents paramètres sont les suivantes :

- Nombre de neurones : 38 ;
- Taux d'apprentissage : 0,04 ;
- Fonction d'activation : ReLU ;
- Algorithme d'optimisation : descente du gradient linéaire ;
- Nombre d'epochs : 350 ;
- Taille du batch : 2000 ;
- *Fold number* : 5 ;
- *Increase ratio* : 1.

Nombre de neurones	38
Taux d'apprentissage	0,04
Taille du batch	2000
Nombre d'epochs	350
Fonction d'activation	ReLU

TABLE 3.35 – Paramètres optimaux obtenus dans l'article [25]

Afin d'éviter de multiplier le nombre de paramètres et de travailler avec des réseaux de neurones relativement simples, le réseau de neurones utilisé ne contient qu'une seule couche cachée. Le nombre de neurones définit plus haut correspond donc au nombre de neurones présents dans cette unique couche cachée.

Le premier paramètre testé est l'algorithme d'optimisation utilisé lors de la mise à jour des poids du réseau de neurones. Comme le montre la figure 3.31, la méthode de descente du gradient linéaire donne des meilleurs résultats que les deux autres méthodes. Les résultats obtenus sont repris dans le tableau 3.36.

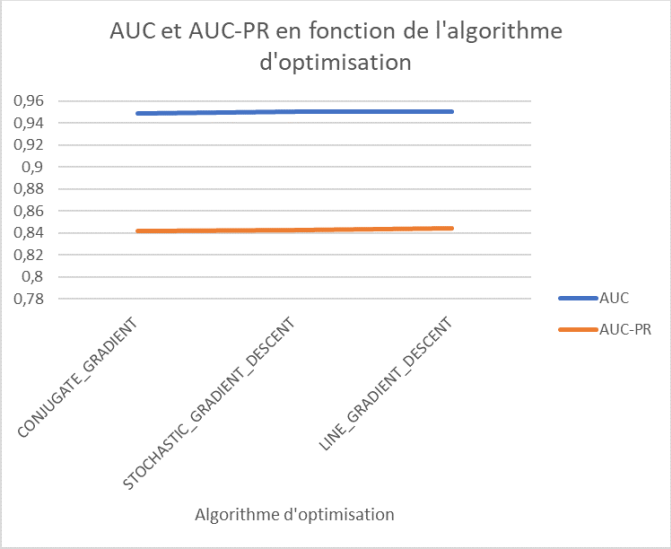

FIGURE 3.31 – Performances en fonction de l'algorithme d'optimsation

AUC	0,9505491, 95% CI [0,94362166 ;0,95747655]
AUC-PR	0,84400409, 95% CI [0,83929975 ;0,84859555]
F-Measure	0,775639, 95% CI [0,75662423 ;0,79465377]

TABLE 3.36 – Indicateurs de performance pour la descente de gradient linéaire

Afin de trouver la valeur optimale de la taille du batch, j'ai testé des valeurs entre 2000 et 10000 par pas de 500. Pour rappel, Alexandre Croix avait identifié la valeur de 2000 comme valeur optimale car l'outil Google Colab, utilisé pour réaliser les tests, ne permettait pas d'utiliser des valeurs supérieures. Comme l'illustre la figure 3.32, plus la taille du batch augmente plus les deux indicateurs AUC et AUC-PR diminuent. La taille du batch optimale est donc de 2000 (résultats dans tableau 3.37).

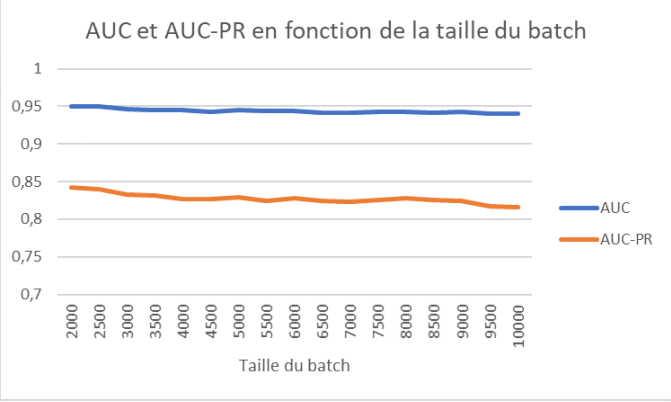

FIGURE 3.32 – Performances en fonction de la taille du batch

Concernant le nombre d'epochs, la valeur optimale déterminée par Alexandre Croix était de 350 qui est la valeur maximale acceptée par Google Colab. Les tests ont donc été effectués sur des valeurs supérieures, de 350 à 700 par incréments de 50 (figure 3.33). Ces tests révèlent qu'au delà d'une valeur de 350 epochs, les améliorations ne semblent plus significatives au regard de

AUC	0,94963211, 95% CI [0,94264531 ;0,95661891]
AUC-PR	0,84261614, 95% CI [0,83789524 ;0,84722461]
F-Measure	0,77369077, 95% CI [0,75461762 ;0,79276392]

TABLE 3.37 – Indicateurs de performance pour une taille de batch de 2000

l'intervalle de confiance à 95% calculé sur les résultats. Pour cette raison, il n'est pas nécessaire d'utiliser des valeurs d'epochs supérieures à 350. Cette valeur sera donc retenue comme valeur optimale. Les résultats obtenus avec ce nombre d'epochs optimal sont visibles dans le tableau 3.38.

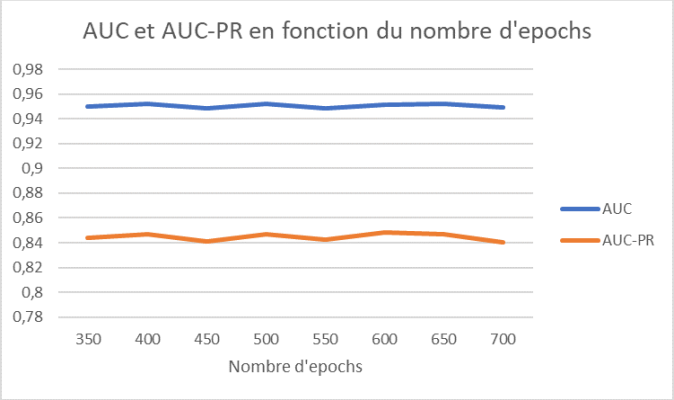

FIGURE 3.33 – Performances en fonction du nombre d'epochs

AUC	0,94957781, 95% CI [0,94258751 ;0,9565681]
AUC-PR	0,8440568, 95% CI [0,8393531 ;0,84864762]
F-Measure	0,77518892, 95% CI [0,75616061 ;0,79421723]

TABLE 3.38 – Indicateurs de performance pour un nombre d'epochs de 350

La série de tests effectués (figure 3.34) pour déterminer la valeur optimale du *fold number* montre, nettement, que plus ce paramètre augmente, plus les performances augmentent, que ce soit pour l'AUC ou l'AUC-PR jusqu'à une valeur de 19 où la croissance semble s'arrêter. Notons que plus cet indice augmente, plus l'ensemble d'entraînement s'agrandit et l'ensemble de test diminue. Concernant l'influence de ce paramètre sur les différents résultats, une étude réalisée sur le sujet [31] a montré que des valeurs comprises entre 10 et 20 réduisent la variance mais augmentent le biais tandis qu'entre 2 et 5, la variance a tendance à augmenter. Les auteurs de cet article recommandent donc d'utiliser une validation croisée à 10 blocs. Cette valeur de 10 sera donc considérée comme la valeur optimale pour la méthode des perceptrons multicouches (tableau 3.39).

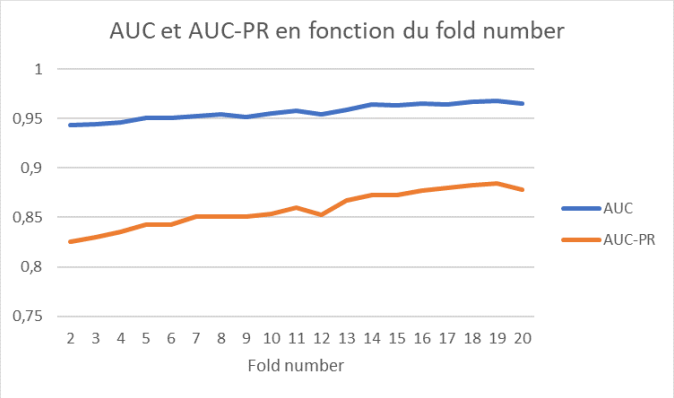

FIGURE 3.34 – Performances en fonction du *fold number*

AUC	0,95532753, 95% CI [0,94872085 ;0,96193421]
AUC-PR	0,85362399, 95% CI [0,84903803 ;0,85809392]
F-Measure	0,78392914, 95% CI [0,76516951 ;0,80268876]

TABLE 3.39 – Indicateurs de performance pour un *fold number* de 10

La tendance observée lors des tests sur le dernier paramètre *increase ratio* est inversée par rapport aux constats effectués avec le *fold number*. Un *increase ratio* de 0 donne une meilleure AUC et AUC-PR que pour une valeur de 10. La valeur optimale conservée est donc nulle. Les performances obtenues avec un *increase ratio* de 0 sont exprimées dans le tableau 3.40.

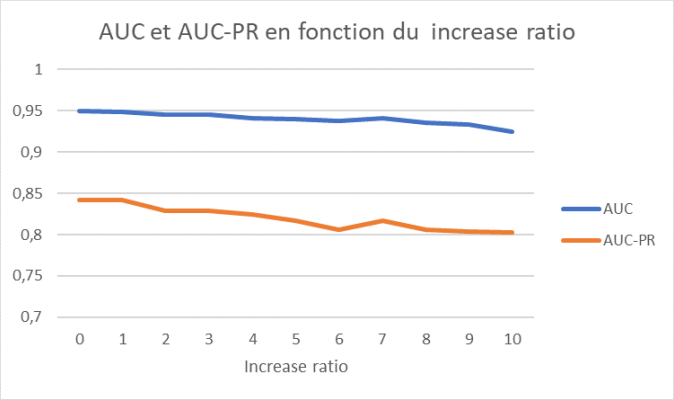

FIGURE 3.35 – Performances en fonction du *increase ratio*

AUC	0,95012226, 95% CI [0,9431671 ;0,95707742]
AUC-PR	0,84228929, 95% CI [0,83756452 ;0,84690176]
F-Measure	0,77112232, 95% CI [0,75197311 ;0,79027154]

TABLE 3.40 – Indicateurs de performance pour un *increase ratio* de 0

La combinaison de tous les paramètres optimaux permet d'obtenir de très bons résultats, qui sont encore meilleurs que ceux obtenus à travers les différents tests, comme en atteste le tableau 3.41. De plus, les performances du perceptron multicouche sont nettement supérieures à celles qui ont été obtenues avec les méthodes précédentes (SVM 3.9, KNN 3.25 et arbres de décision 3.33).

AUC	0,95753632, 95% CI [0,95108496 ;0,96398767]
AUC-PR	0,85940524, 95% CI [0,85489362 ;0,86379894]
F-Measure	0,79676818, 95% CI [0,77842606 ;0,8151103]
Nombre de neurones	38
Taux d'apprentissage	0,04
Algorithme d'optimisation	Descente gradient linéaire
Fonction d'activation	ReLU
Taille du batch	2000
Nombre d'epoch	350
Fold number	10
Increase ratio	0

TABLE 3.41 – Paramètres optimaux et indicateurs de performance pour les MLP

Les résultats (3.42) de la recherche aléatoire des valeurs des différents paramètres, effectuée sur 20 itérations, ne s'améliorent pas, par rapport à ceux repris dans le tableau 3.41. En effet, les intervalles de confiance à 95% des différents indicateurs ne se chevauchent pas.

AUC	0,9335833, 95% CI [0,92410549 ;0,9430611]
AUC-PR	0,80364267, 95% CI [0,79850461 ;0,8086811]
F-Measure	0,73080599, 95% CI [0,70660155 ;0,75501042]
Nombre de neurones	45
Taux d'apprentissage	0,7
Algorithme d'optimisation	Descente gradient linéaire
Fonction d'activation	ReLU
Taille du batch	7500
Nombre d'epoch	650
Fold number	5
Increase ratio	3

TABLE 3.42 – Indicateurs de performance pour les MLP avec paramètres obtenus via la recherche aléatoire

3.7 AdaBoost

En tant que dernière méthode de ce travail, j'ai décidé d'implémenter la méthode ensembliste AdaBoost. En effet, lors de la lecture de mes différents articles, j'ai pu constater que la méthode AdaBoost semblait donner des résultats assez prometteurs. Notamment dans l'article [16] qui présente les résultats de l'utilisation de l'algorithme AdaBoost M1. Mais avant de parler des résultats, qu'est-ce que la méthode AdaBoost ?

L'algorithme AdaBoost (abréviation de *Adaptative Boosting*) a été présenté la première fois par FREUND et SCHAPIRE dans l'article [32] et se base sur la méthode générale du Boosting. Le Boosting est simplement basé sur la combinaison de plusieurs classifieurs (souvent qualifiés d'apprenants faibles - *Weaklearners*) dans le but de former un apprenant fort qui améliore les performances des apprenants faibles. L'algorithme AdaBoost est un algorithme itératif dont la principe se base sur l'apprentissage des erreurs. En effet, lorsqu'une instance de l'ensemble

d'entraînement est mal classée par un classifieur, cette instance se verra attribuer un poids plus élevé dans le classifieur suivant afin que celui-ci puisse classer cette instance correctement. Ce calcul de poids est effectué sur chaque instance d'entraînement et mis à jour après chaque classifieur faisant partie de l'ensemble des apprenants de la méthode AdaBoost.

Afin d'implémenter cette méthode, je me suis basé sur l'article [16] pour l'aspect de l'utilisation d'apprenants autres que des apprenants faibles (la plupart du temps des arbres de décision à un seul étage). Cependant, l'algorithme proposé AdaBoost M1 (figure 3.36) est adapté pour les problèmes multi-classe, ce n'est donc pas le plus pertinent dans le cas présent. C'est pourquoi, j'ai choisi d'implémenter l'algorithme AdaBoost, pour les problèmes binaires, présenté dans l'article [33] dont les différentes étapes vont être exposées dans les lignes qui suivent.

La première étape consiste à attribuer un poids initial à chaque instance de l'ensemble d'entraînement

$$w_0(x_i) = \frac{1}{n} \tag{3.5}$$

n étant le nombre d'instances dans le jeu de données.

Ensuite, en deuxième étape, un classifieur (h_t) est entraîné sur l'ensemble des données d'entraînement avant de calculer l'erreur pondérée. Le calcul de cette valeur s'effectue de la manière suivante :

$$\epsilon_t = \sum_{i=1}^{n} w_t(x_i)[h_t(x_i) \neq y_i] \tag{3.6}$$

y_i étant le label de l'instance i

Lors de la troisième étape, le modèle actuel se voit attribuer un poids qui sera utile pour la prédiction sur de nouvelles instances. Cette valeur appelée *stage* dans l'article se calcule comme suit :

$$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t}) \tag{3.7}$$

Finalement, avant de passer au modèle suivant, le poids de chaque instance est mis à jour. En effet, comme expliqué au début de cette partie, la méthode AdaBoost a pour particularité de porter attention, étape après étape, aux instances qui sont moins bien classifiées. La mise à jour des poids s'effectue de la manière suivante :

$$w_{t+1}(x_i) = \frac{w_t(x_i)}{Z_t} * e^{-\alpha_t * y_i * h_t(x_i)} \tag{3.8}$$

où

$$Z_t = \sum_{i=1}^{n} w_t(x_i) * e^{-\alpha_t * y_i * h_t(x_i)} \tag{3.9}$$

Z_t est un facteur de normalisation afin que les poids des instances répondent à une distribution de probabilité.

Une fois cette mise à jour terminée, le processus peut être réitéré sur le modèle suivant (h_{t+1}) à partir de la deuxième étape et ainsi de suite jusqu'au moment où tous les classifieurs

(T) auront été parcourus. Une fois cela effectué, chaque modèle s'est vu attribuer un α qui va maintenant servir pour évaluer les performances de la méthode sur de nouvelles instances. Chaque classifieur calcule une prédiction sur la nouvelle instance (x_j). Comme nous sommes dans un cas binaire, la prédiction sera de -1 ou 1. Chaque prédiction est ensuite pondérée par la valeur du α du modèle correspondant avant d'être sommée avec celles de tous les autres modèles. La prédiction finale est ensuite déterminée de la manière suivante :

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t * h_t(x)) \tag{3.10}$$

Les poids, mis à jour à chaque itération, sont utilisés afin de donner une importance particulière aux instances qui sont mal classées au fur et à mesure des étapes. Pour certaines méthodes implémentées comme SVM ou KNN, cette distribution peut être directement intégrée comme paramètre. Cependant d'autres méthodes, dans ce cas-ci MLP et DT, ne permettent pas d'intégrer cette nuance dans le modèle. C'est la raison pour laquelle, par souci d'homogénéité, après chaque itération un nouvel ensemble d'entraînement sera rééchantillonné (avec replacement) sur base de l'ensemble et de la distribution de poids de l'itération précédente. Cet ensemble aura toujours la même taille que l'ensemble initial.

Algorithm 1: Adaboost.M1

Input: sequence of m examples $(x_1, y_1), \ldots, (x_m, y_m)$, $x_i \in X$, with labels $y_i \in Y = \{1, \ldots, k\}$

Weak learning algorithm (**Weaklearn**)

Integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i

Do for $t = 1.2, \ldots, T$

1. Call **Weaklearn** and provide it with the distribution D_t
2. Get back a hypothesis $h_t : X \rightarrow Y$
3. Calculate the error of h_t: $\varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$ if $\varepsilon_t > 1/2$ then set $T = T - 1$ and abort loop
4. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$
5. Update distribution D_t : $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t \text{ if } h_t(x_i) = y_i \\ 1 \text{ otherwise} \end{cases}$

   Where Z_t is a normalization constant (Chosen so that D_{t+1} will be a distribution)

Output the final hypothesis: $h_{fin} = \underset{y \in Y}{argmax} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$

FIGURE 3.36 – Algorithme AdaBoost M1 [16]

De manière similaire à ce qui a été réalisé dans l'article [16], l'algorithme AdaBoost sera appliqué avec des classifieurs de différents types, ceux-ci étant simplement les différents modèles précédemment implémentés dans ce travail. Les premiers tests effectués consisteront à évaluer les performances de la méthode AdaBoost dépendant de l'ordre dans lequel les classifieurs interviennent dans la méthode ensembliste. Ensuite, j'appliquerai la méthode AdaBoost avec pour seul changement de ne pas effectuer de rééchantillonnage après chaque itération pour générer

un nouvel ensemble d'entraînement. La méthode s'apparentera donc plutôt à un vote pondéré. Enfin, une dernière série de tests aura pour objectif d'appliquer la méthode AdaBoost de la manière dont elle est habituellement utilisée, c'est-à-dire avec des classifieurs du même type et un rééchantillonage après chaque itération. La combinaison sera donc effectuée individuellement sur chacun des modèles SVM, KNN, arbres de décision et MLP. Je comparerai les résultats dans les cas où les différents modèles sont utilisés avec leurs paramètres optimaux et lorsque d'autres paramètres leurs sont affectés.
La raison pour laquelle tous les essais sont réalisés avec et sans les paramètres optimaux est que, dans la littérature, la méthode AdaBoost est toujours utilisée avec des classifieurs faibles. Les performances de ces apprenants sont légèrement supérieures à celles de classifieurs aléatoires. La méthode AdaBoost crée un apprenant fort à partir de ceux-ci. Les articles qui combinent des apprenants forts avec la méthode AdaBoost sont très peu nombreux. Les essais effectués dans ce travail permettront de comparer ces différentes combinaisons.

3.7.1 AdaBoost avec classifieurs de types différents

3.7.1.1 Paramètres optimaux

Dans cette série de tests, l'algorithme AdaBoost a été appliqué avec 4 classifieurs : SVM, KNN, arbres de décision et MLP. L'évaluation de la méthode a été réalisée, comme pour les test précédents, en utilisant une validation croisée. Le nombre de blocs sélectionné étant de 5. Ce choix permet d'établir un ratio ensemble d'entraînement et de test, communément utilisé, de 80/20. Les résultats obtenus varient de manière assez importante en fonction de l'ordre dans lequel les différents modèles sont considérés. En effet, avec l'ordre : [SVM, MLP, KNN, DT] (tableau 3.44), les résultats sont assez mauvais. Mais lorsque les ordres sont les suivants : [MLP, SVM, KNN, DT] et [MLP, SVM, DT, KNN], on obtient de meilleurs résultats (tableau 3.43). L'ensemble des résultats des tests se trouve en annexe B.

AUC	0,85635556, 95% CI [0,84537119 ;0,86733993]
AUC-PR	0,5389718, 95% CI [0,53258083 ;0,54534999]
F-Measure	0,69785082, 95% CI [0,67692032 ;0,71878132]

TABLE 3.43 – Indicateurs de performance pour [MLP, SVM, KNN, DT] avec paramètres optimaux

AUC	0,43428932, 95% CI [0,42123642 ;0,44734222]
AUC-PR	0,05245236, 95% CI [0,04966913 ;0,05538247]
F-Measure	0,11601428, 95% CI [0,1014172 ;0,13061135]

TABLE 3.44 – Indicateurs de performance pour [SVM, MLP, KNN, DT] avec paramètres optimaux

3.7.1.2 Paramètres non optimaux

Les tests ont été effectués de manière similaire au point précédent à la seule différence que les paramètres des différentes méthodes ont été changés. Cela signifie que les différents classifieurs qui constituent les itérations de l'algorithme AdaBoost sont individuellement moins

performants que précédemment. La méthode AdaBoost est à la base une méthode qui combine des apprenants faibles, ce test a donc pour but d'évaluer l'influence des performances des modèles qui constituent l'ensemble sur les performances de cet ensemble. Ces tests ne permettent pas d'identifier une nette amélioration ou dégradation des résultats de la méthode. Sur base des intervalles de confiance à 95% pour chacun des indicateurs, les combinaisons [KNN, DT, SVM, MLP], [KNN, DT, MLP, SVM], [KNN, MLP, DT, SVM] et [KNN, MLP, SVM, DT] permettent d'obtenir les meilleurs résultats. Les résultats pour [KNN, DT, SVM, MLP] sont exposés dans le tableau 3.45. Tandis que la combinaison [MLP, SVM, DT, KNN] donne les résultats présentés dans le tableau 3.46 qui sont les plus mauvais. Les résultats des autres combinaisons sont disponibles à l'annexe C.

AUC	0,83965601, 95% CI [0,82821089 ;0,85110114]
AUC-PR	0,66418765, 95% CI [0,6581117 ;0,67020973]
F-Measure	0,76233184, 95% CI [0,7429299 ;0,78173378]

TABLE 3.45 – Indicateurs de performance pour [KNN, DT, SVM, MLP] avec paramètres non optimaux

AUC	0,27769195, 95% CI [0,26741751 ;0,2879664]
AUC-PR	0,03545524, 95% CI [0,03316127 ;0,03790168]
F-Measure	0,08087702, 95% CI [0,06844943 ;0,09330461]

TABLE 3.46 – Indicateurs de performance pour [MLP, SVM, DT, KNN] avec paramètres non optimaux

3.7.2 AdaBoost sans rééchantillonnage

Lorsque l'ensemble d'entraînement reste le même pour chaque classifieur et que la méthode est plutôt assimilée à un vote pondéré, les résultats sont beaucoup plus homogènes et dépendent beaucoup moins de l'ordre des différents classifieurs. Notons que l'ensemble n'est pas rééchantillonné mais que le poids de chacune des instances d'entraînement reste mis à jour après chaque itération. Le calcul de l'importance de chaque classifieur repose toujours sur le poids des instances.

3.7.2.1 Paramètres optimaux

En combinant les méthodes avec leurs paramètres optimaux, les meilleurs résultats sont obtenus, à titre indicatif, avec les ordres [KNN, MLP, DT, SVM] (tableau 3.47), [KNN, DT, MLP, SVM] et [DT, MLP, KNN, SVM]. Les moins bons résultats (tableau 3.48) restent assez proches de résultats optimaux et sont obtenus avec l'ordre [KNN, MLP, SVM, DT]. L'ensemble des résultats est disponible dans le tableau de l'annexe D.

AUC	0,82538878, 95% CI [0,81358967 ;0,83718789]
AUC-PR	0,68395974, 95% CI [0,6779746 ;0,68988452]
F-Measure	0,76160602, 95% CI [0,74218373 ;0,78102831]

TABLE 3.47 – Indicateurs de performance pour [KNN, MLP, DT, SVM] sans rééchantillonnage avec paramètres optimaux

AUC	0,82105616, 95% CI [0,80915622 ;0,83295609]
AUC-PR	0,5937682, 95% CI [0,58746236 ;0,60004328]
F-Measure	0,71529412, 95% CI [0,69472441 ;0,73586383]

TABLE 3.48 – Indicateurs de performance pour [KNN, MLP, SVM, DT] sans rééchantillonnage avec paramètres optimaux

3.7.2.2 Paramètres non optimaux

Lorsque les classifieurs s'approchent un peu plus d'apprenants faibles, les résultats sont légèrement meilleurs dans l'ensemble (tableau 3.49) avec la combinaison [KNN, SVM, MLP, DT]. En considérant les intervalles de confiance à 95% pour chaque indicateur, les combinaisons [KNN, DT, SVM, MLP] et [KNN, SVM, DT, MLP] donnent des résultats comparables à ceux de la combinaison [KNN, SVM, MLP, DT]. Cependant avec l'ordre des classifieurs suivant : [DT, KNN, SVM, MLP], les résultats (tableau 3.50) sont très mauvais. Certains indicateurs ne peuvent pas être calculés. Tous les résultats sont repris dans le tableau de l'annexe E.

AUC	0,84031839, 95% CI [0,82889055 ;0,85174622]
AUC-PR	0,66684321, 95% CI [0,66077875 ;0,67285292]
F-Measure	0,76421305, 95% CI [0,74486422 ;0,78356188]

TABLE 3.49 – Indicateurs de performance pour [KNN, SVM, MLP, DT] sans rééchantillonnage avec paramètres non optimaux

AUC	0,5, 95% CI [0,48628902 ;0,51371098]
AUC-PR	NaN
F-Measure	NaN

TABLE 3.50 – Indicateurs de performance pour [DT, KNN, SVM, MLP] sans rééchantillonnage avec paramètres non optimaux

3.7.3 AdaBoost avec classifieurs de même type

La méthode AdaBoost est en général utilisée avec des classifieurs de même type comme les arbres de décision. Deux dernières séries de tests ont été réalisées en combinant plusieurs classifieurs de même type, toujours une première fois avec leurs paramètres optimaux puis avec des paramètres dont le but est de simuler des classifieurs plus faibles.

3.7.3.1 Paramètres optimaux

Lorsque les classifieurs avec leurs paramètres optimaux sont combinés, les meilleurs résultats sont obtenus en associant 4 SVM. La valeur de 4 a été choisie car les tests précédents ont été effectués avec les 4 classifieurs implémentés. J'ai donc décidé de garder cette valeur par soucis de comparaison. Les indicateurs obtenus sont ceux présentés dans le tableau 3.51. Tandis que les moins bons résultats sont obtenus suite à la combinaison de plusieurs KNN (tableau 3.52). L'ensemble des résultats est exposé à l'annexe F.

AUC	0,83529761, 95% CI [0,82374067 ;0,84685456]
AUC-PR	0,59793822, 95% CI [0,59164212 ;0,60420218]
F-Measure	0,72365805, 95% CI [0,7032746 ;0,7440415]

TABLE 3.51 – Indicateurs de performance pour [SVM, SVM, SVM, SVM] avec paramètres optimaux

AUC	0,32862213, 95% CI [0,31725666 ;0,3399876]
AUC-PR	0,03926168, 95% CI [0,0368482 ;0,04182637]
F-Measure	0,08944577, 95% CI [0,07643748 ;0,10245407]

TABLE 3.52 – Indicateurs de performance pour [KNN, KNN, KNN, KNN] avec paramètres optimaux

3.7.3.2 Paramètres non optimaux

Une fois que les classifieurs sont utilisés sans leurs paramètres optimaux, la tendance est inversée par rapport au point précédent. En effet, les meilleurs résultats sont obtenus en combinant plusieurs KNN (tableau 3.53). Contrairement au cas avec les paramètres optimaux, la combinaison de plusieurs SVM donne les moins bons résultats dans le cas présent (tableau 3.54). Les résultats des différentes combinaisons sont visibles dans l'annexe G.

AUC	0,80322045, 95% CI [0,79093582 ;0,81550509]
AUC-PR	0,45770281, 95% CI [0,45132864 ;0,46409086]
F-Measure	0,62280472, 95% CI [0,60071213 ;0,64489731]

TABLE 3.53 – Indicateurs de performance pour [KNN, KNN, KNN, KNN] avec paramètres non optimaux

AUC	0,31173703, 95% CI [0,30071304 ;0,32276102]
AUC-PR	0,04242471, 95% CI [0,03991679 ;0,0450828]
F-Measure	0,09345182, 95% CI [0,08018469 ;0,10671895]

TABLE 3.54 – Indicateurs de performance pour [SVM, SVM, SVM, SVM] avec paramètres non optimaux

3.7.4 Résumé des performances

Suite aux différents tests, dont les résumés des performances sont visibles dans les tableaux 3.55, 3.56 et 3.57, aucune des configurations ne s'est révélée être nettement meilleure qu'une autre. En effet, étant donné les résultats obtenus, et en tenant compte des intervalles de confiance à 95%, presque tous ces intervalles se recouvrent partiellement. Bien qu'extraire la meilleure option ne soit pas possible, certaines d'entre elles peuvent, cependant, être écartées au regard de certains critères. En observant les différentes AUC-PR, 2 groupes peuvent être établis. Le premier groupe, constitué de la méthode avec les paramètres optimaux et les deux options se basant sur les classifieurs de même type, contre-performe par rapport au groupe formé par les 3 autres possibilités. Cette observation est moins flagrante mais toujours présente pour l'indicateur de la *F-Measure*.

Paramètres optimaux	0,85635556, 95% CI [0,84537119 ;0,86733993]
Paramètres non optimaux	0,83965601, 95% CI [0,82821089 ;0,85110114]
Paramètres optimaux sans rééchantillonnage	0,82538878, 95% CI [0,81358967 ;0,83718789]
Paramètres non optimaux sans rééchantillonnage	0,84031839, 95% CI [0,82889055 ;0,85174622]
Paramètres optimaux classifieurs de même type	0,83529761, 95% CI [0,82374067 ;0,84685456]
Paramètres non optimaux classifieurs de même type	0,80322045, 95% CI [0,79093582 ;0,81550509]

TABLE 3.55 – Résumé des AUC obtenues avec la méthode AdaBoost

Paramètres optimaux	0,5389718, 95% CI [0,53258083 ;0,54534999]
Paramètres non optimaux	0,66418765, 95% CI [0,6581117 ;0,67020973]
Paramètres optimaux sans rééchantillonnage	0,68395974, 95% CI [0,6779746 ;0,68988452]
Paramètres non optimaux sans rééchantillonnage	0,66684321, 95% CI [0,66077875 ;0,67285292]
Paramètres optimaux classifieurs de même type	0,59793822, 95% CI [0,59164212 ;0,60420218]
Paramètres non optimaux classifieurs de même type	0,45770281, 95% CI [0,45132864 ;0,46409086]

TABLE 3.56 – Résumé des AUC-PR obtenues avec la méthode AdaBoost

Paramètres optimaux	0,69785082, 95% CI [0,67692032 ;0,71878132]
Paramètres non optimaux	0,76233184, 95% CI [0,7429299 ;0,78173378]
Paramètres optimaux sans rééchantillonnage	0,76160602, 95% CI [0,74218373 ;0,78102831]
Paramètres non optimaux sans rééchantillonnage	0,76421305, 95% CI [0,74486422 ;0,78356188]
Paramètres optimaux classifieurs de même type	0,72365805, 95% CI [0,7032746 ;0,7440415]
Paramètres non optimaux classifieurs de même type	0,62280472, 95% CI [0,60071213 ;0,64489731]

TABLE 3.57 – Résumé des *F-Measure* obtenues avec la méthode AdaBoost

3.8 Comparaison des méthodes

La comparaison des différentes méthodes (tableaux 3.58, 3.59 et 3.60) implémentées et testées au cours des recherches et exposées dans ce travail montre que les réseaux de neurones permettent d'obtenir les meilleurs résultats. Les arbres de décision semblent également montrer des performances très satisfaisantes et légèrement inférieures aux réseaux de neurones. La méthode des K plus proches voisins arrive juste derrière suivie de la méthode ensembliste AdaBoost et enfin des séparateurs à vaste marge. La méthode AdaBoost, bien qu'elle se base sur plusieurs classifieurs qui, individuellement, donnent de très bons résultats, ne semble pas en

mesure de fournir des performances supérieures. Les raisons de cela peuvent être multiples et notamment liées aux données utilisées. Cependant, un autre élément concernant la méthode KNN pourrait expliquer le phénomène. En effet, comme le mentionne l'article [34], les méthodes ensemblistes se basent sur l'instabilité des classifieurs mais les KNN sont assez stables face au rééchantillonnage. La méthode AdaBoost peut donc éventuellement échouer dans sa tentative d'amélioration des résultats. L'article [34] présente plusieurs alternatives afin de *booster* les KNN.

En comparant les résultats obtenus (tableaux 3.1, 3.2 et 3.3) en utilisant uniquement les sorties des détecteurs ou en appliquant de simples méthodes (moyenne et vote majoritaire) directement sur les sorties de ceux-ci, comme cela a été fait dans la section 3.2, avec les résultats obtenus en utilisant des méthodes de Machine Learning (tableaux 3.58, 3.59 et 3.60), les améliorations sont suffisamment significatives pour statuer sur l'utilité de ces dernières.

SVM	0,84320762, 95% CI [0,83185611 ;0,85455912]
KNN	0,91453308, 95% CI [0,90290391 ;0,92616225]
DT	0,91181024, 95% CI [0,90170452 ;0,92191595]
MLP	0,95753632, 95% CI [0,95108496 ;0,96398767]
AdaBoost avec paramètres non optimaux	0,83965601, 95% CI [0,82821089 ;0,85110114]
AdaBoost avec paramètres optimaux sans rééchantillonnage	0,82538878, 95% CI [0,81358967 ;0,83718789]
AdaBoost avec paramètres non optimaux sans rééchantillonnage	0,84031839, 95% CI [0,82889055 ;0,85174622]

TABLE 3.58 – Résumé des performances obtenues avec les différentes méthodes implémentées

SVM	0,6147003, 95% CI [0,60844817 ;0,6209148]
KNN	0,73432813, 95% CI [0,72863229 ;0,73994708]
DT	0,80257995, 95% CI [0,79743169 ;0,80762893]
MLP	0,85940524, 95% CI [0,85489362 ;0,86379894]
AdaBoost avec paramètres non optimaux	0,66418765, 95% CI [0,6581117 ;0,67020973]
AdaBoost avec paramètres optimaux sans rééchantillonnage	0,68395974, 95% CI [0,6779746 ;0,68988452]
AdaBoost avec paramètres non optimaux sans rééchantillonnage	0,66684321, 95% CI [0,66077875 ;0,67285292]

TABLE 3.59 – Résumé des performances obtenues avec les différentes méthodes implémentées

SVM	0,7389946, 95% CI [0,71897604 ;0,75901317]
KNN	0,6408499, 95% CI [0,61209026 ;0,66960953]
DT	0,76723096, 95% CI [0,74554622 ;0,78891569]
MLP	0,79676818, 95% CI [0,77842606 ;0,8151103]
AdaBoost avec paramètres non optimaux	0,76233184, 95% CI [0,7429299 ;0,78173378]
AdaBoost avec paramètres optimaux sans rééchantillonnage	0,76160602, 95% CI [0,74218373 ;0,78102831]
AdaBoost avec paramètres non optimaux sans rééchantillonnage	0,76421305, 95% CI [0,74486422 ;0,78356188]

TABLE 3.60 – Résumé des performances obtenues avec les différentes méthodes implémentées

Un constat intéressant, qui a pu être effectué à travers les différentes méthodes, est que les méthodes des arbres de décision et des réseaux de neurones sont sensibles à une augmentation du nombre de blocs de la validation croisée. Comme mentionné dans la section 2.3, l'augmentation de ce nombre a pour effet de rendre l'ensemble d'entraînement plus important. Les arbres de décision et les réseaux de neurones ont la spécificité de nécessiter d'un grand nombre de données pour bien généraliser, contrairement aux séparateurs à vaste marge qui parviennent à généraliser avec peu de données.

Afin de comparer les différentes méthodes sous un autre aspect, le temps d'exécution[5] de chaque méthode a été mesuré en utilisant les paramètres optimaux mis à part le *fold number* qui a été fixé à 10 pour que les différents chiffres soient comparables entre eux. Les différents temps sont disponibles dans le tableau 3.61. Les différences de temps sont relativement importantes. Les réseaux de neurones qui sont les plus performants en termes de résultats sont également les plus mauvais concernant la durée d'apprentissage après la méthode AdaBoost. Tandis que les arbres de décision qui arrivent juste derrière les réseaux de neurones dans le tableau des résultats sont presque 350 fois plus rapides que ces derniers. La méthode AdaBoost est logiquement la méthode la plus lente étant donné qu'elle dépend de la durée d'entraînement de chaque modèle.

Méthode	Temps en secondes
SVM	206
KNN	17
MLP	2752
DT	8
AdaBoost	5807

TABLE 3.61 – Temps d'exécution des différents modèles

5. Les tests ont été réalisées sur un ordinateur équipé d'un processeur Intel Core i7-8750H cadencé à 2201 MHz, utilisant 16 Go de RAM et fonctionnant sous la version 10 de Windows.

Chapitre 4

Conclusion

Les différents tests menés tout au long de ce travail ont permis d'obtenir des résultats très satisfaisants dans l'ensemble. Les réseaux de neurones sont les plus efficaces concernant la détection d'attaques informatiques, type *WebShell*, sur un réseau. La classification de nouvelles instances est aussi rapide quel que soit le modèle entraîné utilisé, réseaux de neurones, séparateurs à vaste marge ou encore arbres de décision. Si l'objectif est d'intégrer le modèle à un système de détection d'intrusion (IDS), qui serait régulièrement réentrainé avec de nouvelles données, les différences de temps d'exécution pourraient avoir un impact important. Cependant, en considérant la valeur obtenue avec les réseaux de neurones, qui est bien plus élevée que pour les arbres de décision (ratio de l'ordre de 350), ce temps d'apprentissage reste tout de même raisonnable dans l'absolu et est encore loin de certains modèles dont l'entraînement peut s'étendre sur plusieurs jours voire semaines.

Les résultats obtenus ont permis de mettre en lumière certains éléments importants concernant notamment la méthode AdaBoost. En effet, comme déjà exposé plus haut, bien qu'elle fasse l'objet de la combinaison de plusieurs classifieurs performants, la combinaison ne permet pas d'améliorer les résultats obtenus individuellement. Cette contre-performance peut s'expliquer par le fait que les combinaisons ont été, pour la plupart, effectuées sur des classifieurs forts, ou en tout cas dont les performances ne peuvent pas être assimilées à des apprenants faibles, alors que dans la littérature, les combinaisons sont toujours effectuées sur des apprenants faibles. Les tests ont montré que lorsqu'une combinaison de classifieurs forts est envisagée, une méthode proche d'un vote majoritaire pondéré donne de meilleurs résultats. Des recherches futures pourraient donc s'orienter vers l'expérimentation de ces méthodes. Une autre piste d'amélioration concernant la méthode AdaBoost pourrait être d'approfondir les méthodes de *boosting* dédiées aux classifieurs KNN car, comme déjà dit plus haut, les KNN ne sont pas sensibles à un algorithme AdaBoost classique.

D'autres perspectives peuvent être envisagées à la suite de ce travail comme la recherche dans le domaine du Deep Learning avec des réseaux de neurones profonds type récurrents (*Recurrent Neural Network - RNN*), comme les LSTM (*Long Short Term Memory*) ou GRU (*Gated Recurrent Unit*), ou convolutifs (*Convolutional Neural Network - CNN*). Une dernière piste intéressante à approfondir serait de tester les méthodes présentées dans ce travail avec d'autres types de données comme des menaces persistantes avancées (APT).

Bibliographie

[1] Yang Xin et al. "Machine Learning and Deep Learning Methods for Cybersecurity". In : *IEEE Access* 6 (2018), p. 35365-35381. DOI : 10.1109/ACCESS.2018.2836950.

[2] Jonghoon Lee et al. "Cyber Threat Detection Based on Artificial Neural Networks Using Event Profiles". In : *IEEE Access* 7 (2019), p. 165607-165626. DOI : 10.1109/ACCESS.2019.2953095.

[3] Marzia Zaman et Chung-Horng Lung. "Evaluation of machine learning techniques for network intrusion detection". In : *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium* (2018), p. 1-5. DOI : 10.1109/NOMS.2018.8406212.

[4] Farnaz Gharibian et Ali A. Ghorbani. "Comparative Study of Supervised Machine Learning Techniques for Intrusion Detection". In : *Fifth Annual Conference on Communication Networks and Services Research (CNSR 2007)* (2007), p. 350-358. DOI : 10.1109/CNSR.2007.22.

[5] Chih-Fong Tsai et al. "Intrusion detection by machine learning : A review". In : *Expert Systems with Applications* 36.10 (2009), p. 11994-12000. ISSN : 0957-4174. DOI : https://doi.org/10.1016/j.eswa.2009.05.029. URL : https://www.sciencedirect.com/science/article/pii/S0957417409004801.

[6] Hyunjoo Kim et al. "Design of network threat detection and classification based on machine learning on cloud computing". In : *Cluster Computing* 22 (jan. 2019). DOI : 10.1007/s10586-018-1841-8.

[7] Anna L. Buczak et Erhan Guven. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection". In : *IEEE Communications Surveys Tutorials* 18.2 (2016), p. 1153-1176. DOI : 10.1109/COMST.2015.2494502.

[8] Yasir Hamid, M. Sugumaran et Ludovic Journaux. "Machine Learning Techniques for Intrusion Detection : A Comparative Analysis". In : ICIA-16 (2016). DOI : 10.1145/2980258.2980378. URL : https://doi.org/10.1145/2980258.2980378.

[9] Mohammad Almseidin et al. "Evaluation of machine learning algorithms for intrusion detection system". In : *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)* (2017), p. 000277-000282. DOI : 10.1109/SISY.2017.8080566.

[10] Preeti Mishra et al. "A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection". In : *IEEE Communications Surveys Tutorials* 21.1 (2019), p. 686-728. DOI : 10.1109/COMST.2018.2847722.

[11] Kamran Shaukat et al. "Cyber Threat Detection Using Machine Learning Techniques : A Performance Evaluation Perspective". In : *2020 International Conference on Cyber Warfare and Security (ICCWS)* (2020), p. 1-6. DOI : 10.1109/ICCWS48432.2020.9292388.

[12] S Revathi et A Malathi. "A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection". In : *International Journal of Engineering Research & Technology (IJERT)* 2.12 (2013), p. 1848-1853.

[13] Mahdi Zamani et Mahnush Movahedi. "Machine learning techniques for intrusion detection". In : *arXiv preprint arXiv :1312.2177* (2013).

[14] Taqwa Ahmed et al. "Feature Selection Using Information Gain for Improved Structural-Based Alert Correlation". In : *PLOS ONE* 11 (nov. 2016), e0166017. DOI : 10.1371/journal.pone.0166017.

[15] Mahbod Tavallaee et al. "A detailed analysis of the KDD CUP 99 data set". In : (2009), p. 1-6. DOI : 10.1109/CISDA.2009.5356528.

[16] Ployphan Sornsuwit et Saichon Jaiyen. "A New Hybrid Machine Learning for Cybersecurity Threat Detection Based on Adaptive Boosting". In : *Applied Artificial Intelligence* 33 (mars 2019), p. 1-21. DOI : 10.1080/08839514.2019.1582861.

[17] John McHugh. "Testing Intrusion Detection Systems : A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory". In : *ACM Trans. Inf. Syst. Secur.* 3.4 (nov. 2000), p. 262-294. ISSN : 1094-9224. DOI : 10.1145/382912.382923. URL : https://doi.org/10.1145/382912.382923.

[18] *The Curse of Dimensionality and its Cure*. URL : https://medium.com/analytics-vidhya/the-curse-of-dimensionality-and-its-cure-f9891ab72e5c. Dernière consultation le 01/06/22.

[19] Xavier Siebert. "Analyse des donnéees statistiques. Classification supervisée : SVM". In : 2019.

[20] Vandit Jain. *Everything you need to know about "Activation Functions" in Deep learning models*. URL : https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253. Dernière consultation le 15/02/22.

[21] C. Sinclair, L. Pierce et S. Matzner. "An application of machine learning to network intrusion detection". In : (1999), p. 371-377. DOI : 10.1109/CSAC.1999.816048.

[22] J. Kittler et al. "On combining classifiers". In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.3 (1998), p. 226-239. DOI : 10.1109/34.667881.

[23] James B. Fraley et James Cannady. "The promise of machine learning in cybersecurity". In : *SoutheastCon 2017* (2017), p. 1-6. DOI : 10.1109/SECON.2017.7925283.

[24] Alexandre Croix, Thibault Debatty et Wim Mees. "Training a multi-criteria decision system and application to the detection of PHP webshells". In : (2019), p. 1-8. DOI : 10.1109/ICMCIS.2019.8842705.

[25] Alexandre Croix, Thibault Debatty et Wim Mees. "Comparison of a supervised trained neural network classifier and a supervised trained aggregation". In : ().

[26] *Smile - Statistical Machine Intelligence and Learning Engine*. URL : https://haifengl.github.io/quickstart.html. Dernière consultation le 25/03/22.

[27] E. Frank et al. "Weka : A machine learning workbench for data mining." In : *Data Mining and Knowledge Discovery Handbook : A Complete Guide for Practitioners and Researchers*. Sous la dir. d'O. Maimon et L. Rokach. Berlin : Springer, 2005, p. 1305-1314. URL : http://researchcommons.waikato.ac.nz/handle/10289/1497. Dernière consultation le 20/05/22.

[28] Vijay KOTU et Bala DESHPANDE. "Chapter 4 - Classification". In : *Data Science (Second Edition)*. Sous la dir. de Vijay KOTU et Bala DESHPANDE. Second Edition. Morgan Kaufmann, 2019, p. 65-163. ISBN : 978-0-12-814761-0. DOI : https://doi.org/10.1016/B978-0-12-814761-0.00004-6. URL : https://www.sciencedirect.com/science/article/pii/B9780128147610000046.

[29] *WEKA - The workbench for machine learning*. URL : https://waikato.github.io/weka-site/index.html. Dernière consultation le 04/04/22.

[30] *Deeplearning4j Suite Overview*. URL : https://deeplearning4j.konduit.ai/. Dernière consultation le 20/04/22.

[31] Ron KOHAVI. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection". In : 14 (mars 2001).

[32] Yoav FREUND et Robert E SCHAPIRE. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In : *Journal of Computer and System Sciences* 55.1 (1997), p. 119-139. ISSN : 0022-0000. DOI : https://doi.org/10.1006/jcss.1997.1504. URL : https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[33] Tu CHENGSHENG, Liu HUACHENG et Xu BING. "AdaBoost typical Algorithm and its application research". In : *MATEC Web of Conferences* 139 (jan. 2017), p. 00222. DOI : 10.1051/matecconf/201713900222.

[34] Nicolás GARCÍA-PEDRAJAS et Domingo ORTIZ-BOYER. "Boosting k-nearest neighbor classifier by means of input space projection". In : *Expert Systems with Applications* 36.7 (2009), p. 10570-10582. ISSN : 0957-4174. DOI : https://doi.org/10.1016/j.eswa.2009.02.065. URL : https://www.sciencedirect.com/science/article/pii/S0957417409002140.

[35] *C99 WebShell with PHP7 and MySQL Support*. URL : https://github.com/cermmik/C99-WebShell. Dernière consultation le 01/06/22.

Liste des figures

Liste des tableaux

Liste des symboles

AdaBoost Adaptative Boosting

AFDA Applied Functional Data Analysis

AFRL/SNHS Air Force Research Laboratory

ANN Artificial Neural Network

API Application Programming Interface

APT Advanced Persistent Threat

ASCII American Standard Code for Information Interchange

AUC Area Under the Curve

AUC-PR Area Under the Curve Precision Recall

BN Bayesian Network

BPSO Binary Particle Swarm Optimization

CANN Cluster Center And Nearest Neighbor

CFS Correlation-based Feature Selection

CICIDS Canadian Institute for Cybersecurity Intrusion Detection System

CNN Convolutional Neural Network

CSOACN Clustering-based Self-Organized Ant Colony Networks

DAPRA Defense Advanced Resaerch Projects Academy

DL4J Deep Learning for Java

DoS Denial of Service

DT Decision Tree

ERM Ecole Royale Militaire

FN False Negative

FNR False Negative Rate

FNT Flexible Neural Tree

FP False Positive

FPR False Positive Rate

GA Genetical Algorithm

GFR Gradual Feature Removal

GP Genetical Programming

GRU Gated Recurrent Unit

HMM Hidden Markov Models

IDS Intrusion Detection System

IP Internet Protocol

IPDS Indexed Partial Distance Search

IPS Intrusion Prevention System

ISCX Installation Support Center of Expertise

KDD Knowledge Discovery and Data Mining

KNN K Nearest Neighbor

LMDRT Logarithm Marginal Density Ratios Tranfsormation

LSTM Long Short Term Memory

MAE Mean Absolute Error

MARS Multivariate Adaptive Regression Spline

MASFAD Multi-Agent System for APT Detection

MLP Machine Learning

MLP Multilayer Perceptron

NaN Not a Number

NSL-KDD Network Security Laboratory - Knowledge Discovery and Data Mining

PHP Hypertext Preprocessor,

R2L Remote to Local

RBF Radial Basis Function

ReLU Rectified Linear Unit

RF Random Forest

RMSE Root Mean Square Error

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

RWS Roulette Wheel Selection

Smile Statisticel Machine Intelligence and Learning Engine

TanH Tangente hyperbolique

TN True Negative

TNR True Negative Rate

TOS Tournament Selection

TP True Positive

TPR True Positive Rate

U2R User to Root

UNB-CIC University of New Brunswick - Canadian Institute for Cybersecurity

UNSW University of New South Wales

WOWA Weighted Ordered Weighted Averaging

Annexe A

Exemple de WebShell

```php
<?php
/*
****************************************************************************************************
       ________  ________  ________.__           .__  .__                 ________       ____
  ____/   __   \/   __   \/   _____/|  |__   ____ |  | |  |   ___  __    \_____  \     /_   |
_/ ___\____    /\____    /\_____  \ |  |  \_/ __ \|  | |  |   \  \/ /     /  ____/      |   |
\  \___  /    /    /    / /        \|   Y  \  ___/|  |_|  |__  \   /     /       \      |   |
 \___  >/____/    /____/ /_______  /|___|  /\___  >____/____/   \_/ /\  \_______ \ /\  |___|
     \/                          \/      \/     \/                  \/          \/ \/
 
*  c99shell.php v.2.1 (PHP 7) (1.12.2019) Updated by: KaizenLouie for PHP 7, and cermmik for MySQL
*  https://github.com/KaizenLouie/C99Shell-PHP7
*  https://github.com/cermmik/C99-WebShell
****************************************************************************************************
*/

if (!function_exists("getmicrotime"))
{
    function getmicrotime()
    {
        list($usec, $sec) = explode(" ", microtime());
        return ((float)$usec + (float)$sec);
    }
}
error_reporting(5);
@ignore_user_abort(true);
$win = strtolower(substr(PHP_OS, 0, 3)) == "win";
define("starttime", getmicrotime());
if (get_magic_quotes_gpc())
{
    if (!function_exists("strips"))
    {
        function strips(&$arr, $k = "")
        {
            if (is_array($arr))
            {
                foreach ($arr as $k => $v)
                {
                    if (strtoupper($k) != "GLOBALS")
                    {
                        strips($arr["$k"]);
                    }
                }
            }
            else
            {
                $arr = stripslashes($arr);
            }
        }
    }
    strips($GLOBALS);
}
$_REQUEST = array_merge($_COOKIE, $_GET, $_POST);
foreach ($_REQUEST as $k => $v)
{
    if (!isset($$k))
    {
        $$k = $v;
    }
}
$shver = "2.1 [PHP 7 Update] [1.12.2019]";
if (!empty($unset_surl))
{
    setcookie("c99sh_surl");
    $surl = "";
}
elseif (!empty($set_surl))
{
    $surl = $set_surl;
    setcookie("c99sh_surl", $surl);
}

...

?>
```

Figure A.1 – Extrait du script PHP de *WebShell* C99 [35]

Annexe B

AdaBoost avec paramètres optimaux

Ordre des classifieurs	AUC
[DT, KNN, MLP, SVM]	0,6435936, 95% CI [0,6295451 ;0,65764209]
[KNN, DT, MLP, SVM]	0,72613406, 95% CI [0,7126644 ;0,73960372]
[KNN, MLP, DT, SVM]	0,76347102, 95% CI [0,75048666 ;0,77645537]
[KNN, MLP, SVM, DT]	0,78284911, 95% CI [0,77018037 ;0,79551784]
[DT, MLP, KNN, SVM]	0,65311604, 95% CI [0,63910288 ;0,6671292]
[MLP, DT, KNN, SVM]	0,63156053, 95% CI [0,61747833 ;0,64564273]
[MLP, KNN, DT, SVM]	0,72113267, 95% CI [0,70760938 ;0,73465595]
[MLP, KNN, SVM, DT]	0,70120273, 95% CI [0,68749075 ;0,71491471]
[DT, MLP, SVM, KNN]	0,7483816, 95% CI [0,73518249 ;0,76158071]
[MLP, DT, SVM, KNN]	0,75595649, 95% CI [0,74286192 ;0,76905107]
[MLP, SVM, DT, KNN]	0,85090433, 95% CI [0,8397637 ;0,86204495]
[MLP, SVM, KNN, DT]	0,85635556, 95% CI [0,84537119 ;0,86733993]
[DT, KNN, SVM, MLP]	0,75307819, 95% CI [0,73994312 ;0,76621325]
[KNN, DT, SVM, MLP]	0,77678604, 95% CI [0,76401355 ;0,78955854]
[KNN, SVM, DT, MLP]	0,85417155, 95% CI [0,84312387 ;0,86521923]
[KNN, SVM, MLP, DT]	0,8471374, 95% CI [0,83589218 ;0,85838263]
[DT, SVM, KNN, MLP]	0,83693963, 95% CI [0,82542443 ;0,84845484]
[SVM, DT, KNN, MLP]	0,82923501, 95% CI [0,81752795 ;0,84094206]
[SVM, KNN, DT, MLP]	0,58460519, 95% CI [0,57050399 ;0,59870638]
[SVM, KNN, MLP, DT]	0,49295655, 95% CI [0,47930155 ;0,50661156]
[DT, SVM, MLP, KNN]	0,81589041, 95% CI [0,80387412 ;0,8279067]
[SVM, DT, MLP, KNN]	0,82630531, 95% CI [0,81452792 ;0,8380827]
[SVM, MLP, DT, KNN]	0,57422716, 95% CI [0,56014511 ;0,58830921]
[SVM, MLP, KNN, DT]	0,43428932, 95% CI [0,42123642 ;0,44734222]

TABLE B.1 – Indicateur AUC avec AdaBoost en fonction de l'ordre des classifieurs avec leurs paramètres optimaux

Ordre des classifieurs	AUC-PR
[DT, KNN, MLP, SVM]	0,10932057, 95% CI [0,10538732 ;0,11338201]
[KNN, DT, MLP, SVM]	0,14836977, 95% CI [0,14387412 ;0,1529808]
[KNN, MLP, DT, SVM]	0,16709222, 95% CI [0,16236825 ;0,17192543]
[KNN, MLP, SVM, DT]	0,19351887, 95% CI [0,18850889 ;0,19862941]
[DT, MLP, KNN, SVM]	0,10921301, 95% CI [0,10528151 ;0,11327274]
[MLP, DT, KNN, SVM]	0,1068247, 95% CI [0,1029322 ;0,1108462]
[MLP, KNN, DT, SVM]	0,14430813, 95% CI [0,13986519 ;0,14886779]
[MLP, KNN, SVM, DT]	0,13191932, 95% CI [0,12764484 ;0,13631457]
[DT, MLP, SVM, KNN]	0,16415653, 95% CI [0,15946685 ;0,16895641]
[MLP, DT, SVM, KNN]	0,16443523, 95% CI [0,15974227 ;0,16923829]
[MLP, SVM, DT, KNN]	0,52817091, 95% CI [0,5217724 ;0,53456018]
[MLP, SVM, KNN, DT]	0,5389718, 95% CI [0,53258083 ;0,54534999]
[DT, KNN, SVM, MLP]	0,17129978, 95% CI [0,16652758 ;0,17617982]
[KNN, DT, SVM, MLP]	0,18832637, 95% CI [0,18336953 ;0,19338547]
[KNN, SVM, DT, MLP]	0,52557707, 95% CI [0,5191772 ;0,53196855]
[KNN, SVM, MLP, DT]	0,52099846, 95% CI [0,51459661 ;0,52739342]
[DT, SVM, KNN, MLP]	0,48058115, 95% CI [0,47418511 ;0,48698357]
[SVM, DT, KNN, MLP]	0,55387448, 95% CI [0,54749886 ;0,56023242]
[SVM, KNN, DT, MLP]	0,10374408, 95% CI [0,09990282 ;0,10771537]
[SVM, KNN, MLP, DT]	0,06227691, 95% CI [0,05925243 ;0,06544502]
[DT, SVM, MLP, KNN]	0,44526763, 95% CI [0,43891103 ;0,45164219]
[SVM, DT, MLP, KNN]	0,54072629, 95% CI [0,53433682 ;0,54710239]
[SVM, MLP, DT, KNN]	0,09711966, 95% CI [0,09339224 ;0,10097927]
[SVM, MLP, KNN, DT]	0,05245236, 95% CI [0,04966913 ;0,05538247]

TABLE B.2 – Indicateur AUC-PR avec AdaBoost en fonction de l'ordre des classifieurs avec leurs paramètres optimaux

Ordre des classifieurs	F-Measure
[DT, KNN, MLP, SVM]	0,20824168, 95% CI [0,18973333 ;0,22675003]
[KNN, DT, MLP, SVM]	0,27062647, 95% CI [0,25037541 ;0,29087753]
[KNN, MLP, DT, SVM]	0,30078905, 95% CI [0,27988537 ;0,32169273]
[KNN, MLP, SVM, DT]	0,34403461, 95% CI [0,32238106 ;0,36568816]
[DT, MLP, KNN, SVM]	0,21353473, 95% CI [0,19485538 ;0,23221407]
[MLP, DT, KNN, SVM]	0,22773992, 95% CI [0,20862428 ;0,24685557]
[MLP, KNN, DT, SVM]	0,27347987, 95% CI [0,25316219 ;0,29379755]
[MLP, KNN, SVM, DT]	0,25252082, 95% CI [0,23271761 ;0,27232404]
[DT, MLP, SVM, KNN]	0,30357886, 95% CI [0,2826204 ;0,32453732]
[MLP, DT, SVM, KNN]	0,30009794, 95% CI [0,27920798 ;0,32098791]
[MLP, SVM, DT, KNN]	0,68928843, 95% CI [0,66819404 ;0,71038281]
[MLP, SVM, KNN, DT]	0,69785082, 95% CI [0,67692032 ;0,71878132]
[DT, KNN, SVM, MLP]	0,3163017, 95% CI [0,29510489 ;0,33749852]
[KNN, DT, SVM, MLP]	0,33708118, 95% CI [0,31553427 ;0,35862809]
[KNN, SVM, DT, MLP]	0,68834417, 95% CI [0,66723224 ;0,70945611]
[KNN, SVM, MLP, DT]	0,68305042, 95% CI [0,66184196 ;0,70425888]
[DT, SVM, KNN, MLP]	0,62401117, 95% CI [0,60193258 ;0,64608975]
[SVM, DT, KNN, MLP]	0,69726293, 95% CI [0,6763209 ;0,71820495]
[SVM, KNN, DT, MLP]	0,18811776, 95% CI [0,17030427 ;0,20593125]
[SVM, KNN, MLP, DT]	0,13564875, 95% CI [0,12004099 ;0,15125652]
[DT, SVM, MLP, KNN]	0,53926594, 95% CI [0,51654563 ;0,56198625]
[SVM, DT, MLP, KNN]	0,68808347, 95% CI [0,66696671 ;0,70920023]
[SVM, MLP, DT, KNN]	0,18031414, 95% CI [0,16279042 ;0,19783785]
[SVM, MLP, KNN, DT]	0,11601428, 95% CI [0,1014172 ;0,13061135]

TABLE B.3 – Indicateur *F-Measure* avec AdaBoost en fonction de l'ordre des classifieurs avec leurs paramètres optimaux

Annexe C

AdaBoost sans paramètres optimaux

Ordre des classifieurs	AUC
[DT, KNN, MLP, SVM]	0,84993819, 95% CI [0,83877048 ;0,8611059]
[KNN, DT, MLP, SVM]	0,83629176, 95% CI [0,82476003 ;0,84782349]
[KNN, MLP, DT, SVM]	0,83709892, 95% CI [0,82558779 ;0,84861006]
[KNN, MLP, SVM, DT]	0,83707869, 95% CI [0,82556704 ;0,84859034]
[DT, MLP, KNN, SVM]	0,67309558, 95% CI [0,65918214 ;0,68700902]
[MLP, DT, KNN, SVM]	0,54777022, 95% CI [0,53377388 ;0,56176655]
[MLP, KNN, DT, SVM]	0,78411199, 95% CI [0,77146545 ;0,79675852]
[MLP, KNN, SVM, DT]	0,78736324, 95% CI [0,77477482 ;0,79995165]
[DT, MLP, SVM, KNN]	0,73363469, 95% CI [0,72025033 ;0,74701906]
[MLP, DT, SVM, KNN]	0,34061948, 95% CI [0,32902348 ;0,35221548]
[MLP, SVM, DT, KNN]	0,27769195, 95% CI [0,26741751 ;0,2879664]
[MLP, SVM, KNN, DT]	0,56604348, 95% CI [0,55198232 ;0,58010464]
[DT, KNN, SVM, MLP]	0,82595941, 95% CI [0,8141738 ;0,83774501]
[KNN, DT, SVM, MLP]	0,83965601, 95% CI [0,82821089 ;0,85110114]
[KNN, SVM, DT, MLP]	0,44921873, 95% CI [0,43598976 ;0,46244769]
[KNN, SVM, MLP, DT]	0,81905467, 95% CI [0,80710916 ;0,83100018]
[DT, SVM, KNN, MLP]	0,68704593, 95% CI [0,6732233 ;0,70086856]
[SVM, DT, KNN, MLP]	0,59133426, 95% CI [0,57722508 ;0,60544343]
[SVM, KNN, DT, MLP]	0,79000741, 95% CI [0,77746728 ;0,80254754]
[SVM, KNN, MLP, DT]	0,82645258, 95% CI [0,81467869 ;0,83822647]
[DT, SVM, MLP, KNN]	0,75070714, 95% CI [0,73753943 ;0,76387484]
[SVM, DT, MLP, KNN]	0,72491249, 95% CI [0,71142949 ;0,73839548]
[SVM, MLP, DT, KNN]	0,50466576, 95% CI [0,49091965 ;0,51841188]
[SVM, MLP, KNN, DT]	0,5418534, 95% CI [0,52788338 ;0,55582342]

TABLE C.1 – Indicateur AUC avec AdaBoost en fonction de l'ordre des classifieurs sans leurs paramètres optimaux

Ordre des classifieurs	AUC-PR
[DT, KNN, MLP, SVM]	0,29131099, 95% CI [0,28552552 ;0,29716494]
[KNN, DT, MLP, SVM]	0,65694082, 95% CI [0,65083459 ;0,66299556]
[KNN, MLP, DT, SVM]	0,66729785, 95% CI [0,66123538 ;0,67330542]
[KNN, MLP, SVM, DT]	0,65738496, 95% CI [0,65128054 ;0,66343774]
[DT, MLP, KNN, SVM]	0,11558703, 95% CI [0,11155432 ;0,11974587]
[MLP, DT, KNN, SVM]	0,12971709, 95% CI [0,12547384 ;0,13408183]
[MLP, KNN, DT, SVM]	0,46400563, 95% CI [0,45762409 ;0,47039898]
[MLP, KNN, SVM, DT]	0,49182114, 95% CI [0,48541928 ;0,49822568]
[DT, MLP, SVM, KNN]	0,15606714, 95% CI [0,1514748 ;0,16077233]
[MLP, DT, SVM, KNN]	0,04312955, 95% CI [0,04060113 ;0,04580791]
[MLP, SVM, DT, KNN]	0,03545524, 95% CI [0,03316127 ;0,03790168]
[MLP, SVM, KNN, DT]	0,13778138, 95% CI [0,13342569 ;0,14225591]
[DT, KNN, SVM, MLP]	0,2759708, 95% CI [0,27028214 ;0,28173297]
[KNN, DT, SVM, MLP]	0,66418765, 95% CI [0,6581117 ;0,67020973]
[KNN, SVM, DT, MLP]	0,06227537, 95% CI [0,05925093 ;0,06544344]
[KNN, SVM, MLP, DT]	0,63592334, 95% CI [0,62973811 ;0,64206398]
[DT, SVM, KNN, MLP]	0,12059669, 95% CI [0,11648725 ;0,12483062]
[SVM, DT, KNN, MLP]	0,08963913, 95% CI [0,08604681 ;0,0933661]
[SVM, KNN, DT, MLP]	0,40928108, 95% CI [0,40299817 ;0,41559375]
[SVM, KNN, MLP, DT]	0,4017104, 95% CI [0,3954474 ;0,40800566]
[DT, SVM, MLP, KNN]	0,16895754, 95% CI [0,16421205 ;0,17381164]
[SVM, DT, MLP, KNN]	0,15179849, 95% CI [0,14725926 ;0,15645196]
[SVM, MLP, DT, KNN]	0,07606669, 95% CI [0,0727398 ;0,07953268]
[SVM, MLP, KNN, DT]	0,07815543, 95% CI [0,07478581 ;0,08166348]

TABLE C.2 – Indicateur AUC-PR avec AdaBoost en fonction de l'ordre des classifieurs sans leurs paramètres optimaux

Ordre des classifieurs	F-Measure
[DT, KNN, MLP, SVM]	0,46778255, 95% CI [0,44503921 ;0,49052589]
[KNN, DT, MLP, SVM]	0,75674056, 95% CI [0,73718385 ;0,77629728]
[KNN, MLP, DT, SVM]	0,76236429, 95% CI [0,74296327 ;0,78176532]
[KNN, MLP, SVM, DT]	0,75763016, 95% CI [0,73809777 ;0,77716256]
[DT, MLP, KNN, SVM]	0,21155706, 95% CI [0,19294105 ;0,23017307]
[MLP, DT, KNN, SVM]	0,16032726, 95% CI [0,14360302 ;0,17705149]
[MLP, KNN, DT, SVM]	0,60788609, 95% CI [0,58563226 ;0,63013992]
[MLP, KNN, SVM, DT]	0,63247863, 95% CI [0,61050247 ;0,65445479]
[DT, MLP, SVM, KNN]	0,29470582, 95% CI [0,27392479 ;0,31548685]
[MLP, DT, SVM, KNN]	0,09579218, 95% CI [0,0823773 ;0,10920706]
[MLP, SVM, DT, KNN]	0,08087702, 95% CI [0,06844943 ;0,09330461]
[MLP, SVM, KNN, DT]	0,16959929, 95% CI [0,15249349 ;0,18670509]
[DT, KNN, SVM, MLP]	0,45368098, 95% CI [0,43098829 ;0,47637368]
[KNN, DT, SVM, MLP]	0,76233184, 95% CI [0,7429299 ;0,78173378]
[KNN, SVM, DT, MLP]	0,12838683, 95% CI [0,11313894 ;0,14363472]
[KNN, SVM, MLP, DT]	0,73519485, 95% CI [0,71508299 ;0,7553067]
[DT, SVM, KNN, MLP]	0,21978714, 95% CI [0,20091178 ;0,2386625]
[SVM, DT, KNN, MLP]	0,17784749, 95% CI [0,16041788 ;0,1952771]
[SVM, KNN, DT, MLP]	0,54861917, 95% CI [0,52593647 ;0,57130186]
[SVM, KNN, MLP, DT]	0,54317049, 95% CI [0,5204649 ;0,56587608]
[DT, SVM, MLP, KNN]	0,31232279, 95% CI [0,29119852 ;0,33344706]
[SVM, DT, MLP, KNN]	0,27650827, 95% CI [0,25612103 ;0,29689551]
[SVM, MLP, DT, KNN]	0,14651693, 95% CI [0,13039827 ;0,16263559]
[SVM, MLP, KNN, DT]	0,15756132, 95% CI [0,14095469 ;0,17416795]

TABLE C.3 – Indicateur *F-Measure* avec AdaBoost en fonction de l'ordre des classifieurs sans leurs paramètres optimaux

Annexe D

AdaBoost avec paramètres optimaux sans rééchantillonnage

Ordre des classifieurs	AUC
[DT, KNN, MLP, SVM]	0,8263632, 95% CI [0,81458719 ;0,83813922]
[KNN, DT, MLP, SVM]	0,82561407, 95% CI [0,81382029 ;0,83740785]
[KNN, MLP, DT, SVM]	0,82538878, 95% CI [0,81358967 ;0,83718789]
[KNN, MLP, SVM, DT]	0,82105616, 95% CI [0,80915622 ;0,83295609]
[DT, MLP, KNN, SVM]	0,82040487, 95% CI [0,80849004 ;0,83231971]
[MLP, DT, KNN, SVM]	0,83136296, 95% CI [0,81970788 ;0,84301803]
[MLP, KNN, DT, SVM]	0,82489032, 95% CI [0,81307945 ;0,83670118]
[MLP, KNN, SVM, DT]	0,82846802, 95% CI [0,81674242 ;0,84019363]
[DT, MLP, SVM, KNN]	0,85032894, 95% CI [0,83917216 ;0,86148572]
[MLP, DT, SVM, KNN]	0,84626472, 95% CI [0,83499565 ;0,8575338]
[MLP, SVM, DT, KNN]	0,83338357, 95% CI [0,82177855 ;0,84498858]
[MLP, SVM, KNN, DT]	0,84075796, 95% CI [0,82934164 ;0,85217428]
[DT, KNN, SVM, MLP]	0,84995366, 95% CI [0,83878638 ;0,86112094]
[KNN, DT, SVM, MLP]	0,84009274, 95% CI [0,82865901 ;0,85152648]
[KNN, SVM, DT, MLP]	0,84381354, 95% CI [0,83247824 ;0,85514884]
[KNN, SVM, MLP, DT]	0,8482545, 95% CI [0,83704001 ;0,85946898]
[DT, SVM, KNN, MLP]	0,84638586, 95% CI [0,83512009 ;0,85765163]
[SVM, DT, KNN, MLP]	0,8433723, 95% CI [0,83202519 ;0,85471941]
[SVM, KNN, DT, MLP]	0,84367203, 95% CI [0,83233293 ;0,85501112]
[SVM, KNN, MLP, DT]	0,84110118, 95% CI [0,82969388 ;0,85250849]
[DT, SVM, MLP, KNN]	0,84499733, 95% CI [0,83369387 ;0,85630078]
[SVM, DT, MLP, KNN]	0,84381766, 95% CI [0,83248247 ;0,85515285]
[SVM, MLP, DT, KNN]	0,84572858, 95% CI [0,83444492 ;0,85701223]
[SVM, MLP, KNN, DT]	0,84751814, 95% CI [0,83628336 ;0,85875292]

TABLE D.1 – Indicateur AUC avec AdaBoost en fonction de l'ordre des classifieurs avec leurs paramètres optimaux sans rééchantillonnage

Ordre des classifieurs	AUC-PR
[DT, KNN, MLP, SVM]	0,63854028, 95% CI [0,63236418 ;0,64467093]
[KNN, DT, MLP, SVM]	0,68349073, 95% CI [0,67750329 ;0,68941796]
[KNN, MLP, DT, SVM]	0,68395974, 95% CI [0,6779746 ;0,68988452]
[KNN, MLP, SVM, DT]	0,5937682, 95% CI [0,58746236 ;0,60004328]
[DT, MLP, KNN, SVM]	0,67974739, 95% CI [0,67374188 ;0,68569393]
[MLP, DT, KNN, SVM]	0,66009187, 95% CI [0,65399861 ;0,6661326]
[MLP, KNN, DT, SVM]	0,66490355, 95% CI [0,65883068 ;0,67092232]
[MLP, KNN, SVM, DT]	0,60225928, 95% CI [0,59597378 ;0,60851123]
[DT, MLP, SVM, KNN]	0,61521098, 95% CI [0,60896032 ;0,62142385]
[MLP, DT, SVM, KNN]	0,61315555, 95% CI [0,60689905 ;0,61937493]
[MLP, SVM, DT, KNN]	0,60256899, 95% CI [0,59628428 ;0,60882006]
[MLP, SVM, KNN, DT]	0,59986458, 95% CI [0,59357315 ;0,60612325]
[DT, KNN, SVM, MLP]	0,60594331, 95% CI [0,59966725 ;0,61218461]
[KNN, DT, SVM, MLP]	0,60423227, 95% CI [0,59795178 ;0,61047856]
[KNN, SVM, DT, MLP]	0,60949582, 95% CI [0,60322922 ;0,61572651]
[KNN, SVM, MLP, DT]	0,62122171, 95% CI [0,61498878 ;0,62741486]
[DT, SVM, KNN, MLP]	0,60406922, 95% CI [0,59778831 ;0,61031598]
[SVM, DT, KNN, MLP]	0,60936861, 95% CI [0,60310166 ;0,61559968]
[SVM, KNN, DT, MLP]	0,62189189, 95% CI [0,615661 ;0,62808278]
[SVM, KNN, MLP, DT]	0,60839535, 95% CI [0,60212577 ;0,61462935]
[DT, SVM, MLP, KNN]	0,60562885, 95% CI [0,59935197 ;0,61187107]
[SVM, DT, MLP, KNN]	0,61668378, 95% CI [0,61043737 ;0,62289191]
[SVM, MLP, DT, KNN]	0,61599055, 95% CI [0,60974213 ;0,62220092]
[SVM, MLP, KNN, DT]	0,61711312, 95% CI [0,61086796 ;0,62331986]

TABLE D.2 – Indicateur AUC-PR avec AdaBoost en fonction de l'ordre des classifieurs avec leurs paramètres optimaux sans rééchantillonnage

Ordre des classifieurs	F-Measure
[DT, KNN, MLP, SVM]	0,73886883, 95% CI [0,71884715 ;0,75889052]
[KNN, DT, MLP, SVM]	0,76151677, 95% CI [0,74209198 ;0,78094155]
[KNN, MLP, DT, SVM]	0,76160602, 95% CI [0,74218373 ;0,78102831]
[KNN, MLP, SVM, DT]	0,71529412, 95% CI [0,69472441 ;0,73586383]
[DT, MLP, KNN, SVM]	0,75585073, 95% CI [0,7362698 ;0,77543166]
[MLP, DT, KNN, SVM]	0,7540188, 95% CI [0,73438838 ;0,77364923]
[MLP, KNN, DT, SVM]	0,75177743, 95% CI [0,73208711 ;0,77146776]
[MLP, KNN, SVM, DT]	0,72430859, 95% CI [0,70393999 ;0,74467718]
[DT, MLP, SVM, KNN]	0,74280927, 95% CI [0,72288631 ;0,76273223]
[MLP, DT, SVM, KNN]	0,73944851, 95% CI [0,71944121 ;0,7594558]
[MLP, SVM, DT, KNN]	0,72727273, 95% CI [0,70697252 ;0,74757294]
[MLP, SVM, KNN, DT]	0,72937201, 95% CI [0,70912091 ;0,7496231]
[DT, KNN, SVM, MLP]	0,73739612, 95% CI [0,71733808 ;0,75745417]
[KNN, DT, SVM, MLP]	0,73165203, 95% CI [0,71145492 ;0,75184913]
[KNN, SVM, DT, MLP]	0,73604061, 95% CI [0,71594935 ;0,75613186]
[KNN, SVM, MLP, DT]	0,74492099, 95% CI [0,72505181 ;0,76479018]
[DT, SVM, KNN, MLP]	0,73431837, 95% CI [0,71418528 ;0,75445147]
[SVM, DT, KNN, MLP]	0,73615819, 95% CI [0,71606981 ;0,75624658]
[SVM, KNN, DT, MLP]	0,74321622, 95% CI [0,72330358 ;0,76312887]
[SVM, KNN, MLP, DT]	0,73460119, 95% CI [0,71447494 ;0,75472744]
[DT, SVM, MLP, KNN]	0,73445378, 95% CI [0,71432396 ;0,7545836]
[SVM, DT, MLP, KNN]	0,74034091, 95% CI [0,72035586 ;0,76032596]
[SVM, MLP, DT, KNN]	0,74088726, 95% CI [0,72091588 ;0,76085864]
[SVM, MLP, KNN, DT]	0,74253521, 95% CI [0,72260531 ;0,76246511]

TABLE D.3 – Indicateur *F-Measure* avec AdaBoost en fonction de l'ordre des classifieurs avec leurs paramètres optimaux sans rééchantillonnage

Annexe E

AdaBoost sans paramètres optimaux et sans rééchantillonnage

Ordre des classifieurs	AUC
[DT, KNN, MLP, SVM]	0,5, 95% CI [0,48628902 ;0,51371098]
[KNN, DT, MLP, SVM]	0,83629703, 95% CI [0,82476543 ;0,84782863]
[KNN, MLP, DT, SVM]	0,83475205, 95% CI [0,82318133 ;0,84632276]
[KNN, MLP, SVM, DT]	0,83565201, 95% CI [0,82410403 ;0,84719999]
[DT, MLP, KNN, SVM]	0,58473954, 95% CI [0,57063815 ;0,59884093]
[MLP, DT, KNN, SVM]	0,78525807, 95% CI [0,77263187 ;0,79788428]
[MLP, KNN, DT, SVM]	0,78869629, 95% CI [0,7761321 ;0,80126048]
[MLP, KNN, SVM, DT]	0,78169806, 95% CI [0,76900926 ;0,79438685]
[DT, MLP, SVM, KNN]	0,5165249, 95% CI [0,50269651 ;0,53035329]
[MLP, DT, SVM, KNN]	0,78495027, 95% CI [0,77231859 ;0,79758195]
[MLP, SVM, DT, KNN]	0,77817281, 95% CI [0,76542364 ;0,79092199]
[MLP, SVM, KNN, DT]	0,79128894, 95% CI [0,77877255 ;0,80380533]
[DT, KNN, SVM, MLP]	0,5, 95% CI [0,48628902 ;0,51371098]
[KNN, DT, SVM, MLP]	0,83654473, 95% CI [0,82501945 ;0,84807002]
[KNN, SVM, DT, MLP]	0,83534465, 95% CI [0,82378889 ;0,84690041]
[KNN, SVM, MLP, DT]	0,84031839, 95% CI [0,82889055 ;0,85174622]
[DT, SVM, KNN, MLP]	0,5, 95% CI [0,48628902 ;0,51371098]
[SVM, DT, KNN, MLP]	0,70160331, 95% CI [0,68789474 ;0,71531188]
[SVM, KNN, DT, MLP]	0,70236919, 95% CI [0,68866717 ;0,7160712]
[SVM, KNN, MLP, DT]	0,70136354, 95% CI [0,68765293 ;0,71507416]
[DT, SVM, MLP, KNN]	0,5, 95% CI [0,48628902 ;0,51371098]
[SVM, DT, MLP, KNN]	0,70212796, 95% CI [0,68842388 ;0,71583204]
[SVM, MLP, DT, KNN]	0,70183671, 95% CI [0,68813013 ;0,71554328]
[SVM, MLP, KNN, DT]	0,70125274, 95% CI [0,68754119 ;0,7149643]

TABLE E.1 – Indicateur AUC avec AdaBoost en fonction de l'ordre des classifieurs sans leurs paramètres optimaux sans rééchantillonnage

Ordre des classifieurs	AUC-PR
[DT, KNN, MLP, SVM]	NaN
[KNN, DT, MLP, SVM]	0,65181467, 95% CI [0,64568797 ;0,65789155]
[KNN, MLP, DT, SVM]	0,65099739, 95% CI [0,6448675 ;0,65707773]
[KNN, MLP, SVM, DT]	0,64941245, 95% CI [0,64327643 ;0,65549945]
[DT, MLP, KNN, SVM]	0,35198885, 95% CI [0,34589604 ;0,35813023]
[MLP, DT, KNN, SVM]	0,62660338, 95% CI [0,6203872 ;0,63277802]
[MLP, KNN, DT, SVM]	0,62491422, 95% CI [0,6186927 ;0,63109476]
[MLP, KNN, SVM, DT]	0,62689433, 95% CI [0,62067908 ;0,63306794]
[DT, MLP, SVM, KNN]	0,0684811, 95% CI [0,06531592 ;0,07178787]
[MLP, DT, SVM, KNN]	0,62540183, 95% CI [0,61918184 ;0,63158067]
[MLP, SVM, DT, KNN]	0,621208, 95% CI [0,61497504 ;0,6274012]
[MLP, SVM, KNN, DT]	0,62631374, 95% CI [0,62009664 ;0,6324894]
[DT, KNN, SVM, MLP]	NaN
[KNN, DT, SVM, MLP]	0,65669109, 95% CI [0,65058385 ;0,66274693]
[KNN, SVM, DT, MLP]	0,65782429, 95% CI [0,65172167 ;0,66387513]
[KNN, SVM, MLP, DT]	0,66684321, 95% CI [0,66077875 ;0,67285292]
[DT, SVM, KNN, MLP]	NaN
[SVM, DT, KNN, MLP]	0,53414948, 95% CI [0,52775477 ;0,54053299]
[SVM, KNN, DT, MLP]	0,53442212, 95% CI [0,52802761 ;0,54080534]
[SVM, KNN, MLP, DT]	0,53462827, 95% CI [0,52823391 ;0,54101128]
[DT, SVM, MLP, KNN]	NaN
[SVM, DT, MLP, KNN]	0,53480336, 95% CI [0,52840912 ;0,54118617]
[SVM, MLP, DT, KNN]	0,5335341, 95% CI [0,52713895 ;0,53991824]
[SVM, MLP, KNN, DT]	0,5316783, 95% CI [0,52528191 ;0,5380643]

TABLE E.2 – Indicateur AUC-PR avec AdaBoost en fonction de l'ordre des classifieurs sans leurs paramètres optimaux sans rééchantillonnage

Ordre des classifieurs	F-Measure
[DT, KNN, MLP, SVM]	NaN
[KNN, DT, MLP, SVM]	0,75440167, 95% CI [0,73478155 ;0,77402179]
[KNN, MLP, DT, SVM]	0,75306794, 95% CI [0,73341202 ;0,77272387]
[KNN, MLP, SVM, DT]	0,75275872, 95% CI [0,73309454 ;0,77242291]
[DT, MLP, KNN, SVM]	0,2870077, 95% CI [0,26638826 ;0,30762713]
[MLP, DT, KNN, SVM]	0,7025066, 95% CI [0,68166881 ;0,72334438]
[MLP, KNN, DT, SVM]	0,70499837, 95% CI [0,68421127 ;0,72578547]
[MLP, KNN, SVM, DT]	0,69880319, 95% CI [0,67789145 ;0,71971493]
[DT, MLP, SVM, KNN]	0,06454971, 95% CI [0,05334901 ;0,07575041]
[MLP, DT, SVM, KNN]	0,70161556, 95% CI [0,68075984 ;0,72247129]
[MLP, SVM, DT, KNN]	0,69315526, 95% CI [0,67213383 ;0,71417669]
[MLP, SVM, KNN, DT]	0,70806242, 95% CI [0,68733867 ;0,72878617]
[DT, KNN, SVM, MLP]	NaN
[KNN, DT, SVM, MLP]	0,75688623, 95% CI [0,73733348 ;0,77643897]
[KNN, SVM, DT, MLP]	0,75661058, 95% CI [0,73705031 ;0,77617084]
[KNN, SVM, MLP, DT]	0,76421305, 95% CI [0,74486422 ;0,78356188]
[DT, SVM, KNN, MLP]	NaN
[SVM, DT, KNN, MLP]	0,56582422, 95% CI [0,54323188 ;0,58841656]
[SVM, KNN, DT, MLP]	0,56701807, 95% CI [0,54443303 ;0,58960312]
[SVM, KNN, MLP, DT]	0,56549641, 95% CI [0,5429021 ;0,58809073]
[DT, SVM, MLP, KNN]	NaN
[SVM, DT, MLP, KNN]	0,56669179, 95% CI [0,54410473 ;0,58927884]
[SVM, MLP, DT, KNN]	0,56593821, 95% CI [0,54334656 ;0,58852985]
[SVM, MLP, KNN, DT]	0,56443105, 95% CI [0,54183037 ;0,58703173]

TABLE E.3 – Indicateur *F-Measure* avec AdaBoost en fonction de l'ordre des classifieurs sans leurs paramètres optimaux sans rééchantillonnage

Annexe F

AdaBoost avec classifieurs de même type et leurs paramètres optimaux

Ordre des classifieurs	AUC
[DT, DT, DT, DT]	0,62041342, 95% CI [0,6063107 ;0,63451614]
[KNN, KNN, KNN, KNN]	0,32862213, 95% CI [0,31725666 ;0,3399876]
[SVM, SVM, SVM, SVM]	0,83529761, 95% CI [0,82374067 ;0,84685456]
[MLP, MLP, MLP, MLP]	0,35430141, 95% CI [0,34245473 ;0,36614809]

TABLE F.1 – Indicateur AUC avec AdaBoost avec des classifieurs de même type avec leurs paramètres optimaux

Ordre des classifieurs	AUC-PR
[DT, DT, DT, DT]	0,09804204, 95% CI [0,09429845 ;0,10191751]
[KNN, KNN, KNN, KNN]	0,03926168, 95% CI [0,0368482 ;0,04182637]
[SVM, SVM, SVM, SVM]	0,59793822, 95% CI [0,59164212 ;0,60420218]
[MLP, MLP, MLP, MLP]	0,04712484, 95% CI [0,04448368 ;0,04991462]

TABLE F.2 – Indicateur AUC-PR avec AdaBoost avec des classifieurs de même type avec leurs paramètres optimaux

Ordre des classifieurs	F-Measure
[DT, DT, DT, DT]	0,20898985, 95% CI [0,19045704 ;0,22752266]
[KNN, KNN, KNN, KNN]	0,08944577, 95% CI [0,07643748 ;0,10245407]
[SVM, SVM, SVM, SVM]	0,72365805, 95% CI [0,7032746 ;0,7440415]
[MLP, MLP, MLP, MLP]	0,10212063, 95% CI [0,08831827 ;0,11592299]

TABLE F.3 – Indicateur *F-Measure* avec AdaBoost avec des classifieurs de même type avec leurs paramètres optimaux

Annexe G

AdaBoost avec classifieurs de même type et sans leurs paramètres optimaux

Ordre des classifieurs	AUC
[MLP, MLP, MLP, MLP]	0,37112592, 95% CI [0,35898878 ;0,38326307]
[KNN, KNN, KNN, KNN]	0,80322045, 95% CI [0,79093582 ;0,81550509]
[SVM, SVM, SVM, SVM]	0,31173703, 95% CI [0,30071304 ;0,32276102]
[DT, DT, DT, DT]	0,36641548, 95% CI [0,35435769 ;0,37847328]

TABLE G.1 – Indicateur AUC avec AdaBoost avec des classifieurs de même type sans leurs paramètres optimaux

Ordre des classifieurs	AUC-PR
[MLP, MLP, MLP, MLP]	0,06586015, 95% CI [0,06275343 ;0,06910934]
[KNN, KNN, KNN, KNN]	0,45770281, 95% CI [0,45132864 ;0,46409086]
[SVM, SVM, SVM, SVM]	0,04242471, 95% CI [0,03991679 ;0,0450828]
[DT, DT, DT, DT]	0,05079353, 95% CI [0,0480536 ;0,05368088]

TABLE G.2 – Indicateur AUC-PR avec AdaBoost avec des classifieurs de même type sans leurs paramètres optimaux

Ordre des classifieurs	F-Measure
[MLP, MLP, MLP, MLP]	0,10043239, 95% CI [0,08673173 ;0,11413305]
[KNN, KNN, KNN, KNN]	0,62280472, 95% CI [0,60071213 ;0,64489731]
[SVM, SVM, SVM, SVM]	0,09345182, 95% CI [0,08018469 ;0,10671895]
[DT, DT, DT, DT]	0,10452404, 95% CI [0,09057891 ;0,11846918]

TABLE G.3 – Indicateur *F-Measure* avec AdaBoost avec des classifieurs de même type sans leurs paramètres optimaux