Le Machine Learning confronté aux contraintes opérationnelles des systèmes de détection

Anaël Bonneton[1,2] et Antoine Husson[1]
prenom.nom@ssi.gouv.fr

[1] ANSSI
[2] INRIA – ENS Paris

Résumé. Les systèmes de détection d'intrusion, reposant traditionnellement sur des signatures, n'ont pas échappé à l'attrait récent des techniques de Machine Learning. Si les résultats présentés dans les articles de recherche académique sont souvent excellents, les experts en sécurité ont cependant encore de nombreuses réticences concernant l'utilisation du Machine Learning dans les systèmes de détection d'intrusion. Ils redoutent généralement une inadéquation de ces techniques aux contraintes opérationnelles, notamment à cause d'un niveau d'expertise requis important, ou d'un grand nombre de faux positifs.
Dans cet article, nous montrons que le Machine Learning peut être compatible avec les contraintes opérationnelles des systèmes de détection. Nous expliquons comment construire un modèle de détection et présentons de bonnes pratiques pour le valider avant sa mise en production. La méthodologie est illustrée par un cas d'étude sur la détection de fichiers PDF malveillants et nous proposons un outil libre, SecuML, pour la mettre en œuvre.

1 Introduction

Les systèmes de détection d'intrusion reposent traditionnellement sur des signatures générées manuellement par des experts en sécurité. Or, le *Big Data*, l'*intelligence artificielle*, le *Machine Learning* (apprentissage automatique en français) ou le *Deep Learning* (apprentissage profond en français) sont souvent présentés comme des technologies pouvant révolutionner les systèmes de détection d'intrusion [18, 23]. En effet, ces méthodes induisent des règles de détection automatiquement à partir de données, et leurs capacités de généralisation leur permettent de détecter des événements malveillants encore inconnus.

De nombreux papiers de recherche sur l'application de l'apprentissage automatique à la détection d'intrusion ont été publiés et présentent souvent des résultats exceptionnels (détection de PDF malveillants [6, 12, 20, 21], de fichiers exécutables malveillants [11], ou de botnets [2, 3]). Cependant,

d'un point de vue opérationnel, il reste encore de nombreuses réticences sur l'utilisation de l'apprentissage automatique en production :

- Un système de détection reposant sur des méthodes d'apprentissage automatique peut-il réaliser son traitement en temps réel ?
- Le taux de faux positifs des méthodes d'apprentissage automatique, souvent présumé trop important, est-il acceptable pour qu'elles soient mises en production ?
- L'apprentissage automatique n'est pas le cœur de métier d'un expert en sécurité, pourtant c'est sur lui que repose la mise en place d'un système de détection. Comment peut-il avoir confiance en ces méthodes pour les mettre en production ?
- Les alertes générées par un tel système de détection sont-elles suffisamment interprétables pour permettre leur exploitation en production ?

Dans ce papier, nous répondons aux interrogations mentionnées ci-dessus en apportant des solutions pour que les méthodes d'apprentissage automatique ne soient pas incompatibles avec les contraintes opérationnelles, et qu'elles puissent ainsi être intégrées dans les systèmes de détection en complément d'autres méthodes, comme les signatures. Tout d'abord, nous présentons brièvement le problème de la détection d'intrusion et nous donnons une vue générale de l'apprentissage automatique dans ce contexte (cf. section 2). Puis, nous présentons un cas d'étude : la détection de fichiers PDF malveillants (cf. section 3). Cet exemple sera ensuite utilisé pour illustrer de bonnes pratiques pour apprendre et valider un modèle de détection, et mettre en avant des écueils parfois omis dans les publications académiques (cf. sections 4 et 5). Enfin, nous présentons l'outil libre SecuML (cf. section 6) permettant notamment de construire un modèle de détection et de le valider en amont de sa mise en production avec les bonnes pratiques exposées dans le papier.

2 Détection d'intrusion et apprentissage automatique

Le rôle d'un système de détection d'intrusion est de détecter les événements malveillants à travers les activités réseau et système qu'il analyse. Un événement suspect peut être la récupération d'un fichier malveillant joint à un courriel ou la visite d'un site Internet corrompu par exemple. L'*administrateur du système de détection* est notamment responsable de la mise en place des méthodes de détection, et de les faire évoluer au cours du temps. L'*opérateur de sécurité* analyse et qualifie les alertes afin

de permettre la prise des mesures nécessaires pour traiter les éventuels incidents de sécurité. Ce travail peut se révéler coûteux.

Les systèmes de détection d'intrusion reposent traditionnellement sur des signatures : des règles de détection construites par un expert à la suite d'une analyse approfondie d'événements malveillants. Cette approche est efficace contre les menaces qui ont déjà été observées et pour lesquelles une signature a été générée, mais elle est souvent inefficace pour détecter de nouvelles menaces. De plus, de simples variations de la menace, comme le polymorphisme [5], peuvent suffire à rendre la signature inefficace. Les signatures sont omniprésentes dans les systèmes de détection actuels, mais des méthodes de détection avec de l'apprentissage automatique sont considérées en complément afin de mieux détecter les nouvelles menaces. Dans cette section, nous présentons les deux grandes catégories de méthodes d'apprentissage automatique pouvant compléter l'approche par signatures : la *détection d'anomalies* et l'*apprentissage supervisé*.

2.1 Détection d'anomalies

La détection d'anomalies est la première méthode d'apprentissage automatique qui a été appliquée à la détection d'intrusion [7]. Cette approche nécessite uniquement des données non malveillantes pour construire le modèle de détection. Le modèle va ensuite générer une alerte dès qu'un événement diffère trop du comportement normal induit des données bénignes fournies initialement.

Les méthodes de détection d'anomalies sont très attrayantes, car elles permettent de détecter des menaces inconnues. Elles n'ont aucun a priori sur ce qu'est un événement malveillant et sont donc enclines à détecter de nouvelles menaces. Par ailleurs, leur mise en production est souvent présentée comme très simple : il suffit d'avoir un jeu de données bénignes dépourvu d'activités malveillantes. Obtenir un tel jeu de données n'est cependant pas facile en pratique, car il n'existe aucun moyen simple de s'assurer de l'absence d'activité malveillante. Si le jeu de données supposé sain comporte des activités malveillantes, cela peut fausser l'apprentissage du modèle et empêcher la détection de certaines menaces. Ces systèmes de détection sont simples en principe, mais rarement en pratique, et leur mise en production peut être très complexe.

De plus, les méthodes de détection d'anomalies lèvent des alertes pour des événements anormaux qui ne sont pas nécessairement malveillants. Par exemple, un ratio émission/réception anormal sur HTTPS peut être le signe d'une exfiltration de données, mais peut aussi être causé par l'utilisation de certains réseaux sociaux ; des sites web populaires peuvent

être à l'origine d'échanges de données semblant anormalement importants, sans pour autant être malveillants ; et de simples erreurs de configuration peuvent aussi entraîner des comportements déclenchant de fausses alertes. Ainsi, ces méthodes de détection souffrent souvent d'un fort taux de faux positifs.

Enfin, la détection d'anomalies offre peu de possibilités pour prendre en compte des connaissances d'experts. En effet, les experts ne peuvent pas guider ces modèles en fournissant des exemples d'événements malveillants comme ils ne prennent en compte que des événements bénins.

2.2 Apprentissage supervisé

L'apprentissage supervisé répond à ce besoin d'intégration des connaissances d'experts. En effet, un modèle de détection supervisé est construit à partir de données labélisées fournies par l'expert : des événements bénins, mais aussi des événements malveillants pour guider le modèle de détection. L'algorithme d'apprentissage va automatiquement chercher les points permettant de caractériser chacune des classes ou de les discriminer pour construire le modèle de détection. Une fois le modèle de détection appris sur un jeu de données d'apprentissage, il peut être appliqué automatiquement pour détecter des événements malveillants.

Grâce à l'apprentissage supervisé, l'opérateur de sécurité supervisant le système de détection peut facilement participer à l'amélioration du modèle de détection à partir des alertes qu'il analyse. En effet, les fausses alertes peuvent être réinjectées pour corriger le modèle de détection et ainsi éviter de générer les mêmes fausses alertes dans le futur. Les vraies alertes peuvent elles aussi être réinjectées dans le modèle pour lui faire suivre l'évolution de la menace. Ainsi, les experts en sécurité ne donnent pas le contrôle du système de détection à un modèle automatique, mais ils le supervisent activement pour améliorer ses performances au cours du temps [16].

Enfin, l'apprentissage supervisé est guidé par des exemples malveillants fournis par l'expert, ce qui permet de réduire le taux de faux positifs par rapport à la détection d'anomalies. Les méthodes supervisées sont donc à privilégier lorsque des données labélisées sont disponibles pour entraîner le modèle de détection. Ces méthodes doivent cependant être appliquées en prenant en compte les contraintes opérationnelles des systèmes de détection. Le modèle de détection doit pouvoir traiter les données en temps réel, et le taux de faux positifs doit rester en dessous d'un certain seuil pour éviter que l'opérateur de sécurité ne soit submergé par les fausses alertes. Enfin, l'administrateur doit avoir confiance dans le modèle

pour le mettre en production, et l'opérateur doit pouvoir comprendre les alertes générées. Dans la suite du papier, nous donnons une méthodologie pour que l'apprentissage automatique réponde à ces contraintes, et qu'il puisse ainsi être intégré dans les systèmes de détection pour mieux détecter les nouvelles menaces.

3 Étude de cas : détection de fichiers PDF malveillants

Dans cette section, nous présentons le problème de la détection de fichiers PDF malveillants basée sur de l'apprentissage automatique. La suite de l'article s'appuiera sur ce cas d'étude pour illustrer les bonnes pratiques et mettre en avant les écueils à éviter lorsqu'on souhaite utiliser l'apprentissage automatique pour construire un modèle de détection.

3.1 Problème

Le format PDF est un format ouvert de description de document, créé par la société Adobe en 1993, visant à préserver la mise en forme indépendamment du logiciel de lecture ou du système d'exploitation utilisé. Il est constitué entre autres d'un certain nombre de métadonnées comme l'auteur, la date de production, ainsi que des objets de différents types référencés dans une table appelée Xref. Ces objets peuvent être notamment du texte, des images, de la vidéo ou encore du code JavaScript. Sa richesse et la disponibilité des lecteurs sur différentes plateformes en font un format très utilisé dans la plupart des organisations pour créer et échanger des documents électroniques. En revanche, le volume des spécifications associées (plus de 1 300 pages disponibles publiquement) implique une complexité logicielle importante, amplifiée par des dépendances avec de nombreuses bibliothèques tierces. Aussi, ces logiciels sont souvent sujets à des vulnérabilités, qui rendent le format PDF d'autant plus attrayant pour les attaquants.

Dans la majorité des cas, les fichiers PDF malveillants sont forgés par l'attaquant pour exploiter une vulnérabilité, afin d'exécuter du code et de compromettre la machine de la victime (par exemple du code JavaScript exploitant une vulnérabilité du moteur JavaScript inclus dans le lecteur, ou une police TTF exploitant une vulnérabilité de l'OS). Parmi les éléments que l'on pourra chercher à repérer pour détecter des fichiers PDF malveillants, on peut noter :

– des caractéristiques typiques liées au déclenchement de la vulnérabilité (utilisation de la fonction OpenAction, code JavaScript, etc.) ;

– la présence d'une charge malveillante (un shellcode, etc.) ;
– des fonctions pour leurrer la détection (obfuscation par chiffrement, encodages multiples, dissimulation d'objets, etc.) ;
– le caractère plus ou moins réaliste des fichiers (malformations, faible nombre de pages et/ou d'objets, etc.).

L'apprentissage supervisé est à privilégier dans le contexte des systèmes de détection d'intrusion (cf. section 2), et il est facile de se procurer des fichiers PDF bénins et malveillants à l'aide de Contagio[3], VirusTotal[4], ou du moteur de recherche de Google par exemple. Nous avons donc choisi l'apprentissage supervisé pour construire un modèle de détection de fichiers PDF malveillants. La section suivante expose les grandes étapes de l'utilisation de l'apprentissage supervisé, et décrit les estimateurs de performance permettant d'évaluer convenablement un modèle de détection.

3.2 Apprentissage supervisé

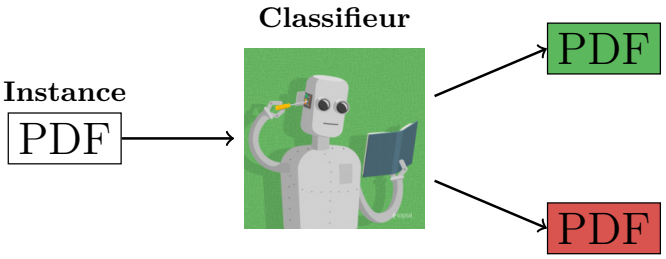


Fig. 1. Classifieur binaire pour la détection d'intrusion

Deux phases : apprentissage et prédiction L'apprentissage supervisé peut être utilisé pour la détection d'intrusion via un classifieur binaire (cf. figure 1). Le classifieur prend en entrée une *instance*, un fichier PDF par exemple, et retourne en sortie le label prédit, bénin ou malveillant (en vert et rouge sur la figure). Ce classifieur peut aussi être appelé *modèle de détection* dans le cadre de la détection d'intrusion.

L'apprentissage supervisé comporte deux grandes étapes :

1. apprentissage du classifieur à partir de données labélisées ;
2. utilisation du classifieur pour détecter des instances malveillantes.

[3] `http://contagiodump.blogspot.fr/`
[4] `https://www.virustotal.com/`

Au cours de la première étape, le classifieur est construit à partir de données labélisées, c'est-à-dire d'un ensemble de fichiers PDF malveillants et bénins dont les labels sont connus (cf. figure 2). Ces données labélisées utilisées pour apprendre le modèle sont appelées *données d'apprentissage*. Le classifieur est créé par un algorithme d'apprentissage automatique cherchant les points communs des instances partageant le même label, et les points discriminants des instances de labels différents.

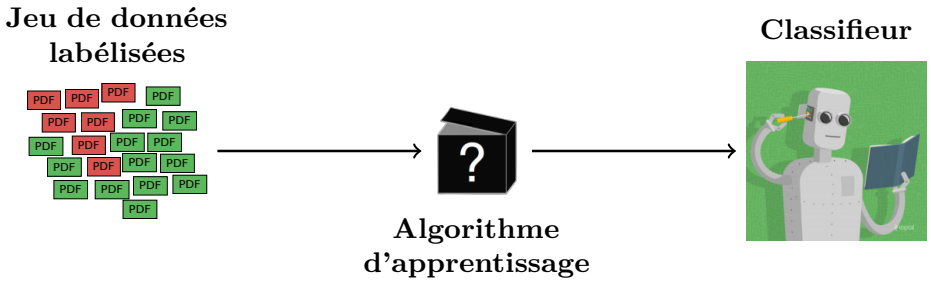


Fig. 2. Phase d'apprentissage du classifieur

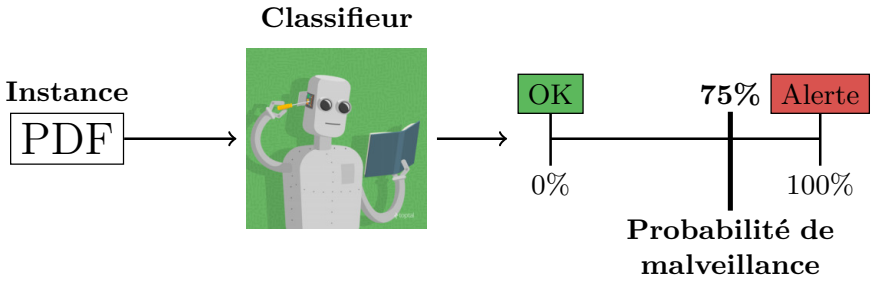


Fig. 3. Phase de prédiction

Une fois que le classifieur est entraîné à partir des données d'apprentissage, il peut être utilisé pour prédire le label d'un fichier PDF. En pratique, la plupart des classifieurs ne prédisent pas simplement une valeur binaire (bénin vs malveillant), mais plutôt une probabilité de malveillance (cf. figure 3). Une alerte est alors générée seulement si la probabilité de malveillance est supérieure au seuil de détection fixé par l'administrateur du système de détection. Dans l'exemple de la figure 3, une alerte sera levée pour le fichier PDF considéré uniquement si le seuil de détection est inférieur à 75%. La probabilité de malveillance prédite par le modèle de détection permet de classer les alertes en fonction de la confiance du

modèle, et donc de définir la priorité des alertes que doit traiter l'opérateur supervisant le système de détection.

Estimateurs de performance Un modèle de détection n'est pas parfait, il peut commettre des erreurs de prédiction. Il est primordial de le valider, c'est-à-dire de mesurer la pertinence des alertes générées, avant de le mettre en production.

L'estimateur de performance le plus connu est le taux d'erreur de classification qui est égal au pourcentage d'instances mal classifiées. Cependant, dans le cas de la détection d'intrusion les données sont en général très asymétriques (avec une faible proportion d'instances malveillantes), et le taux d'erreur n'est pas capable d'estimer correctement la performance d'un classifieur dans cette situation. Voici un exemple présentant les limites du taux d'erreur de classification. On considère 100 instances : 2 malveillantes et 98 bénignes. Dans cette situation, un modèle de détection niais prédisant toujours bénin aura un taux d'erreur de classification de seulement 2% alors qu'il n'est pas capable de détecter la moindre instance malveillante.

Afin d'analyser correctement les performances d'un modèle de détection, la première étape consiste à écrire la matrice de confusion qui prend en compte les deux types d'erreurs possibles : les faux positifs, c'est-à-dire les fausses alertes levées pour des instances bénignes, et les faux négatifs, c'est-à-dire des instances malveillantes non détectées. La figure 4 explique le contenu d'une matrice de confusion.

		Label prédit	
		`Malveillant`	`Bénin`
Vrai label	`Malveillant`	Vrai Positif (VP)	Faux Négatif (FN)
	`Bénin`	Faux Positif (FP)	Vrai Négatif (VN)

Vrai	la prédiction est vraie (label prédit = vrai label)
Faux	la prédiction est fausse (label prédit ≠ vrai label)
Positif	la prédiction est `Malveillant`
Négatif	la prédiction est `Bénin`

Fig. 4. Matrice de confusion

La matrice de confusion permet d'exprimer des indicateurs de performance tels que le taux de détection ou le taux de faux positifs :

$$\text{Taux de faux positifs} = \frac{FP}{FP + VN}$$

$$\text{Taux de détection} = \frac{VP}{VP + FN}.$$

Un modèle de détection doit être évalué avec ces deux estimateurs de performance pris ensemble. En effet, le taux de faux positifs doit être faible pour que l'opérateur de sécurité supervisant le système de détection ne soit pas submergé par les fausses alertes. Le taux de détection doit quant à lui être élevé pour éviter que trop de menaces restent non détectées.

Le seuil de détection détermine la sensibilité de la détection : baisser ce seuil augmente le taux de détection, mais aussi le taux de fausses alertes. Il est ainsi fixé par l'administrateur du système de détection en fonction du compromis souhaité entre taux de détection et taux de faux positifs.

Les estimateurs de performance que nous venons de présenter dépendent de la valeur du seuil de détection. Un autre estimateur de performance, qui a l'avantage d'être indépendant de ce seuil, est souvent utilisé en détection : la fonction d'efficacité du récepteur, plus fréquemment désignée sous le terme de courbe ROC[5] [9]. Cette courbe représente le taux de détection en fonction du taux de faux positifs pour divers seuils de détection (cf. figure 5). Pour un seuil de 100%, les taux de détection et de fausses alertes sont nuls, et pour un seuil de 0% ils sont tout deux à 100%. Un modèle de détection est d'autant plus performant que sa courbe est proche du coin supérieur gauche : un fort taux de détection pour un faible taux de fausses alertes. L'aire sous la courbe ROC, appelée AUC[6], est souvent calculée pour estimer la performance d'un modèle de détection indépendamment du seuil de détection, et sa valeur doit être proche de 1.

Un classifieur prédisant de manière aléatoire la probabilité de malveillance a pour courbe ROC la droite rouge représentée sur la figure 5. Ainsi, la courbe ROC d'un classifieur doit toujours être au-dessus de cette droite (sinon un classifieur aléatoire a de meilleures performances...), et l'AUC est au minimum de 0.5. La courbe ROC est non seulement un estimateur de performance, mais elle permet aussi à l'administrateur du système de détection de choisir la valeur du seuil de détection en fonction du taux de détection souhaité ou du taux de fausses alertes toléré.

Méthode générique Notre présentation de l'apprentissage supervisé s'est appuyée sur l'exemple de la détection de fichiers PDF malveillants, mais l'instance peut aussi représenter un fichier DOC, le trafic associé à une adresse IP ou une page web par exemple. Les algorithmes d'apprentissage automatique ne prennent pas en entrée les instances à l'état brut, mais une

[5] Receiver Operating Characteristic.
[6] Area Under the Curve.

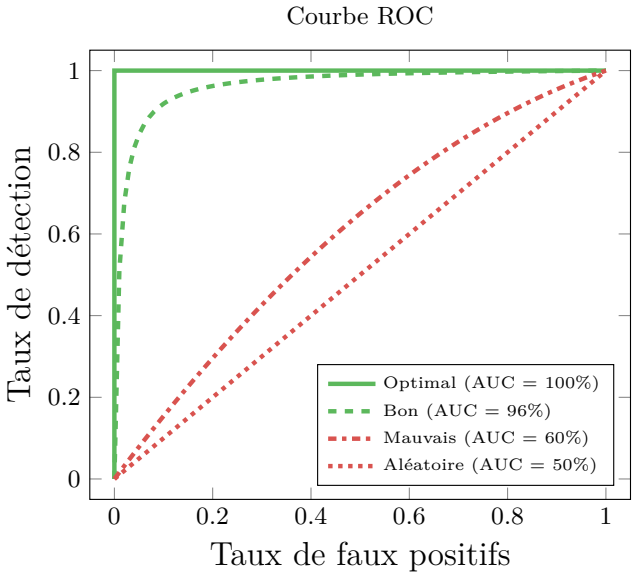


Fig. 5. Explication de la courbe ROC

représentation sous forme de vecteurs d'attributs numériques[7] de taille fixe. Grâce à cette représentation des instances, les algorithmes d'apprentissage automatique sont génériques et peuvent être facilement appliqués à divers problèmes de détection d'intrusion. L'étape d'extraction d'attributs est quant à elle spécifique à chaque problème de détection.

Dans la section suivante, nous présentons les principales étapes qu'il faut réaliser pour créer un modèle de détection supervisé en illustrant nos propos avec le cas de la détection de fichiers PDF malveillants.

4 Comment construire un modèle de détection ?

La première étape avant de construire un modèle de détection avec de l'apprentissage automatique est de définir la cible, c'est-à-dire ce qu'on souhaite détecter. Cette étape préliminaire a été effectuée pour le problème de la détection de fichiers PDF malveillants dans la section 3.1.

Ensuite, pour créer un modèle de détection il faut :

- recueillir des données d'apprentissage contenant des instances bénignes et malveillantes correspondant à la cible ;
- définir les attributs à extraire pour représenter les instances sous forme de vecteurs numériques ;

[7] Des attributs catégoriels peuvent aussi être utilisés avec certains modèles de classification, mais nous allons les omettre pour simplifier les explications dans le cadre de ce papier.

– choisir un type de modèle de classification adapté aux contraintes opérationnelles.

Nous décrivons maintenant ces trois étapes en donnant des conseils génériques et des exemples à partir du cas d'étude sur les fichiers PDF.

4.1 Obtenir des données d'apprentissage

La construction d'un modèle de détection supervisé nécessite des données d'apprentissage pour lesquelles les labels, bénins ou malveillants, sont connus. Ces données d'apprentissage doivent contenir des instances bénignes et malveillantes correspondant à la cible de détection. Or, obtenir des données d'apprentissage est souvent coûteux, car associer un label à des instances nécessite les connaissances d'un expert en sécurité. Il existe des jeux de données labélisées publics pour certains problèmes de détection (projet Malicia, KDD99, kyoto2006, ou Contagio par exemple), mais pour les autres problèmes une phase d'annotation de données doit précéder la construction du modèle. Cette étape peut être fastidieuse, mais l'efficacité du modèle sera directement liée à la composition du jeu d'apprentissage, c'est pourquoi cette étape ne doit pas être négligée.

Il y a quelques grands principes à respecter lors de la construction du jeu d'apprentissage. Tout d'abord, il doit comporter un nombre suffisant d'instances pour que le modèle de détection soit capable de généraliser correctement les comportements bénin et malveillant. Typiquement, il semble très difficile d'apprendre un modèle si les données d'apprentissage contiennent moins de quelques centaines d'instances pour chaque label. Par ailleurs, il faut faire attention à ne pas avoir un jeu de données d'apprentissage trop déséquilibré avec un label représentant une très faible proportion des données. Il est impossible de définir des règles générales concernant le nombre minimal d'instances nécessaires pour apprendre un modèle, ou le degré de déséquilibre acceptable, car ces valeurs dépendent du problème de détection considéré et des instances constituant le jeu d'apprentissage.

Dans le cas de la détection de fichiers PDF malveillants, il est aisé d'obtenir un jeu de données labélisées, car ce type de fichiers est populaire et fréquemment utilisé pour propager du code malveillant. Les expériences présentées dans ce papier s'appuient sur deux jeux de données : Contagio (9 000 fichiers bénins et 11 101 malveillants) et webPdf (2 078 fichiers bénins et 767 malveillants). Contagio est un jeu de données public utilisé dans de nombreux travaux académiques, et nous avons construit webPdf

à partir de fichiers bénins issus du moteur de recherche de Google et de fichiers malveillants obtenus sur la plateforme VirusTotal.

4.2 Extraire des attributs discriminants

Cette étape consiste à traiter les données afin de les représenter sous forme de vecteurs d'attributs numériques de taille fixe exploitables par les algorithmes d'apprentissage. Les attributs sont des caractéristiques numériques extraites des données qui vont permettre la prise de décision du classifieur : plus elles sont discriminantes pour la prise de décision, plus le classifieur sera efficace. Il faut donc avoir une bonne connaissance du format et du contenu des données considérées, et avoir bien défini la cible de détection pour extraire des attributs discriminants.

La phase d'extraction des attributs est spécifique à chaque problème de détection, mais des techniques usuelles d'extraction d'attributs numériques peuvent souvent être appliquées. Nous présentons quelques méthodes classiques d'extraction d'attributs en donnant des exemples avec le cas des fichiers PDF.

Les fichiers PDF contiennent deux types d'informations pouvant servir à générer des attributs : les métadonnées (l'auteur ou la date de création par exemple), et la liste de ses objets. Certaines informations sont déjà numériques, ou une simple transformation peut les rendre numériques. Par exemple, la taille du fichier est numérique et les dates de création et de modification peuvent être transformées en horodatages. Cependant, d'autres informations, comme l'auteur ou les objets, ne sont pas numériques, et ne peuvent donc pas être exploitées directement par les algorithmes d'apprentissage automatique. Par ailleurs, chaque fichier PDF a un nombre variable d'objets : comment représenter ces informations sous forme d'un vecteur de taille fixe ?

Chaînes de caractères Les informations discriminantes peuvent se présenter sous forme de chaînes de caractères. La méthode d'extraction que nous avons utilisée consiste à transformer une chaîne de caractères en un vecteur d'attributs où chaque attribut correspond à une famille de caractères (les majuscules, les minuscules, ou les chiffres par exemple). La valeur d'un attribut est déterminée par le nombre d'occurrences, ou la proportion, de cette famille dans la chaîne de caractères.

L'auteur d'un fichier PDF est une chaîne de caractères que nous avons transformée en 7 attributs numériques : taille de la chaîne, nombre de minuscules, majuscules, chiffres, caractères « point », caractères « espace », et autres caractères spéciaux.

Listes numériques de taille variable Des listes numériques de taille variable peuvent aussi contenir des informations pertinentes pour la détection. Comme les vecteurs d'attributs utilisés par les algorithmes d'apprentissage doivent tous avoir la même taille, il est nécessaire de traiter ces listes pour en extraire un nombre fixe d'attributs numériques. La manière habituelle de transformer ces listes est de calculer des indicateurs statistiques comme la moyenne et la variance, ou encore la taille de la liste et les extremums.

Les fichiers PDF sont constitués d'objets dont on peut récupérer la taille. À partir de la liste des tailles des objets, il est possible d'extraire la moyenne et la variance de la taille des objets, ainsi que les tailles minimales et maximales des objets.

Caractéristiques catégorielles Certaines caractéristiques que l'on souhaite utiliser sont catégorielles et ne peuvent pas être exploitées directement. Pour transformer une caractéristique catégorielle en valeurs numériques, l'idée naturelle est de faire correspondre un numéro à chaque catégorie. Cependant, cette idée n'est pas utilisable, car les algorithmes d'apprentissage reposent sur des distances entre les attributs, et que deux catégories avec des numéros proches seraient considérées proches. À l'inverse, deux catégories avec des numéros éloignés seraient considérées très différentes par l'algorithme d'apprentissage.

Une technique souvent utilisée est de transformer une caractéristique catégorielle en un vecteur d'attributs binaires. Pour chacune des catégories, ou seulement les plus fréquentes, on crée un attribut binaire ou numérique qui retranscrit l'appartenance à cette catégorie.

Le type des objets PDF est une bonne manière d'illustrer ce cas de figure. Parmi les types d'objets, on retrouve par exemple *Image*, *Text* et *Police*. Ces trois catégories, et il y en a beaucoup d'autres, ne sont pas ordonnables, et il n'est pas possible de parler de proximité ou de distance entre différents types. C'est pourquoi nous avons adopté une extraction par comptage, où la caractéristique de type est découpée en plusieurs attributs comptant le type des objets. Dans cet exemple, on utiliserait 3 attributs qui représenteraient le nombre d'objets de chacun des types précédemment cités.

La phase d'extraction d'attributs peut être séparée en deux étapes : l'extraction d'informations discriminantes, puis la transformation de ces informations en un nombre fixe d'attributs numériques. L'extraction d'informations nécessite une bonne connaissance de la cible de détection alors que la transformation de ces informations peut suivre des techniques usuelles.

Pour la détection de fichiers PDF malveillants, nous avons extrait 120 attributs numériques, similaires à ceux présentés dans les travaux de Smutz et Stavrou [19, 20], que nous pouvons regrouper en trois catégories :

- métadonnées du fichier (par exemple l'auteur, ou la date de création) ;
- structure du fichier (par exemple le nombre d'objets, ou la taille moyenne des objets) ;
- objets et des mots-clefs utilisés dans le fichier (par exemple les types d'objets, ou le nombre d'objets images).

4.3 Sélection du type de modèle de classification

Toute l'intelligence de l'apprentissage supervisé repose dans l'optimisation des paramètres du modèle de classification à partir des données d'apprentissage pour lesquelles les labels sont connus. Cette phase d'optimisation est théoriquement complexe, mais la grande popularité de l'apprentissage automatique a graduellement simplifié son utilisation avec l'émergence de nombreuses bibliothèques dédiées (scikit-learn en python, Spark, Mahout ou Weka en java, ou Vowpal Wabbit en C++ par exemple), et de solutions en ligne comme Google Cloud ML, Microsoft Azure ou Amazon Machine Learning. Il est ainsi facile d'apprendre divers modèles de classification à partir de données d'apprentissage.

Les réseaux de neurones sont tellement populaires [8], notamment grâce à leurs résultats extraordinaires en vision par ordinateur, que l'amalgame entre apprentissage profond et apprentissage automatique est souvent fait. Or, les réseaux de neurones ne sont qu'un type de modèles de classification, et il en existe beaucoup d'autres : arbres de décision, forêts aléatoires (ou *Random Forests*), k plus proches voisins, analyses discriminantes linéaires ou quadratiques, régressions logistiques, machines à vecteurs de support (ou *SVM* pour *Support Vector Machine*) ou classification naïve bayésienne pour ne citer que quelques exemples.

Il faut donc choisir un type de modèle de classification adapté au problème de détection considéré, et répondant aux contraintes opérationnelles des systèmes de détection mentionnées dans la section 2. Les modèles de classification linéaires, comme la régression logistique ou les machines à vecteurs de support, sont adaptés à la détection d'intrusion et nous détaillons maintenant comment ils répondent aux contraintes opérationnelles.

Prédictions rapides La plupart des modèles de classification ont une phase d'apprentissage très coûteuse en temps de calcul qui est réalisée hors

ligne, mais l'application du modèle à de nouvelles données est généralement extrêmement rapide. C'est le cas des modèles de classification linéaires dont l'application est en $O(d)$ où d est le nombre d'attributs décrivant chaque instance.

En revanche, certains modèles, qualifiés de paresseux, sont à éviter pour la détection d'intrusion. En effet, ils n'ont pas de phase d'apprentissage, et l'intégralité des données d'apprentissage est donc considérée lors de la phase de prédiction. Par exemple, les k plus proches voisins est un modèle paresseux non adapté à la détection d'intrusion. Pour prédire le label d'une nouvelle instance, il faut chercher ses k plus proches voisins parmi toutes les données d'apprentissage, et ensuite le label prédit est celui le plus représenté. Les prédictions de ce modèle de classification ont une complexité temporelle en $O(nd)$ qui dépend du nombre d'attributs d, mais aussi du nombre de données d'apprentissage n. Or, en apprentissage automatique, il est souhaitable que n soit grand, et n augmente au cours du temps lorsque de nouvelles instances sont réinjectées dans le modèle suite à l'analyse des alertes générées. Le temps de prédiction de ce modèle de classification est beaucoup trop important pour pouvoir être utilisé dans un système de détection.

Mise à jour périodique du modèle Lorsqu'un système de détection est en production, l'opérateur de sécurité assure sa supervision en analysant les alertes générées. Leurs analyses ont pour but premier de repérer les fausses alertes, mais aussi de prendre les mesures nécessaires dans le cas d'un incident de sécurité. Cependant, leurs analyses, aussi bien les faux que les vrais positifs, peuvent être intégrées au modèle de détection afin d'améliorer ses capacités de détection. Il est donc primordial que le modèle puisse être mis à jour périodiquement sans reconsidérer l'intégralité des données d'apprentissage utilisées précédemment.

Les modèles linéaires répondent parfaitement à cette contrainte. Ils peuvent être appris en mode *batch* initialement, mais peuvent aussi être mis à jour de manière incrémentale en ne prenant en compte que les nouvelles données labélisées.

Modèle interprétable Les modèles linéaires ont le grand avantage d'être interprétables. Ainsi, l'administrateur du système de détection peut interpréter le modèle de détection, c'est-à-dire comprendre comment il prend ses décisions, avant de le mettre en production afin de vérifier sa cohérence. Ensuite, l'opérateur supervisant le système de détection peut connaître les principales causes responsables de la génération d'une alerte.

Les modèles linéaires associent un coefficient à chaque attribut qui permet de comprendre le fonctionnement du modèle. Plus la valeur absolue du coefficient d'un attribut est importante, plus l'attribut influence la prédiction. Les attributs ayant un coefficient nul n'ont aucune influence sur les prédictions, ils ont été considérés inutiles pour distinguer les instances malveillantes des bénignes au cours de la phase d'apprentissage. Les attributs ayant un coefficient négatif discriminent les instances bénignes (plus ces attributs sont élevés, plus la probabilité de malveillance sera faible), alors que les attributs avec un coefficient positif discriminent les instances malveillantes (plus ces attributs sont élevés, plus la probabilité de malveillance sera élevée).

La simplicité des modèles de classification linéaires les rend interprétables. À l'inverse, certains modèles de classification, comme les réseaux de neurones, sont tellement complexes qu'il est extrêmement difficile, même pour un expert en apprentissage profond, de les interpréter. Il est donc préférable d'éviter leur utilisation dans le cadre de la détection d'intrusion.

Nous avons montré que les modèles de classification linéaires répondent parfaitement aux contraintes opérationnelles de la détection d'intrusion. Cependant, ces modèles très simples peuvent s'avérer insuffisants pour discriminer les instances malveillantes des bénignes de certains problèmes de détection, et des modèles plus complexes, comme les forêts aléatoires ou les réseaux de neurones, doivent être utilisés en faisant un compromis sur l'interprétabilité, ou le coût de la mise à jour du modèle. En revanche, nous conseillons de toujours faire de premiers tests avec des modèles linéaires avant de s'orienter vers des modèles plus complexes. Nous allons maintenant expliquer comment valider un modèle de détection, et comment savoir si un modèle linéaire est suffisant ou si un modèle plus complexe est nécessaire.

5 Comment valider un modèle de détection ?

La validation d'un modèle de détection avant sa mise en production nécessite de répondre aux questions suivantes :

1. Est-ce que le modèle de détection a appris quelque chose à partir des données d'apprentissage ? Ou bien se comporte-t-il comme un classifieur aléatoire ?
2. Est-ce que le modèle de détection a de bonnes capacités de généralisation ? Peut-il prédire correctement le label de données non utilisées au cours de sa phase d'apprentissage ?

3. Est-ce que le modèle de détection construit est cohérent avec la cible de détection ?

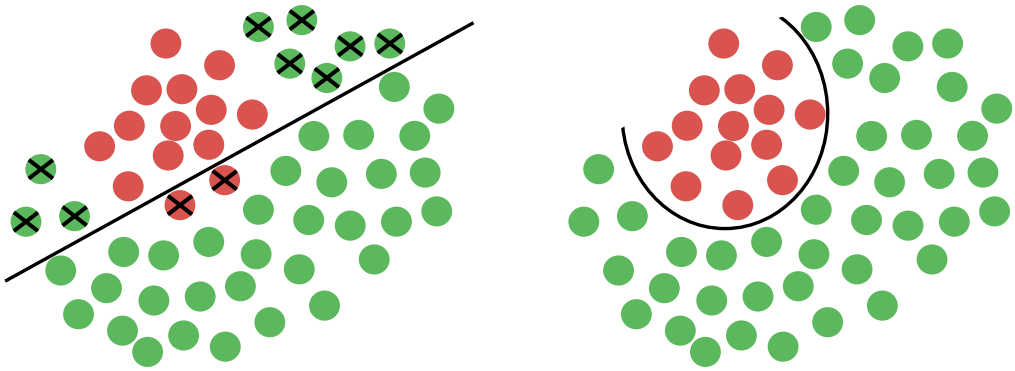
Pour répondre aux deux premières questions, on évalue les performances du modèle sur les données d'apprentissage et sur un jeu de données de validation indépendant. Le jeu de données de validation doit lui aussi être entièrement labélisé afin de vérifier la pertinence des labels prédits avec les vrais labels. Une méthode classique pour construire les jeux d'apprentissage et de validation à partir d'un jeu de données labélisées est de prendre 90% des données pour l'apprentissage, et les 10% restants pour la phase de validation. Il est préférable de ne pas sélectionner les instances de validation de manière aléatoire, mais de prendre les instances les plus récentes pour la validation afin d'avoir des conditions de validation les plus réalistes possible. Enfin, une fois que les deux premières phases de validation ont été effectuées avec succès, on répond à la dernière question en interprétant le modèle de détection pour comprendre son fonctionnement avant sa mise en production.

Nous allons illustrer ces trois étapes avec le cas d'étude sur la détection de fichiers PDF malveillants. Nous utilisons le jeu de données labélisées Contagio pour mettre en place le modèle. Nous l'avons divisé en un jeu d'apprentissage, appelé Contagio_90%, et un jeu de validation, appelé Contagio_10%. Nous apprenons un modèle de détection linéaire, une régression logistique, comme recommandé dans la section 4.3, avec les attributs présentés dans la section 4.2. Pour illustrer ce qui peut se produire au moment de la mise en production, nous utilisons les données de webPdf comme données de production. webPdf contient des PDF plus proches d'un environnement de production que Contagio, qui sont aussi labélisés, ce qui permet de valider le modèle très facilement.

5.1 Validation sur les données d'apprentissage

Nous tenons à préciser que les performances du modèle de détection sur ses données d'apprentissage ne sont pas une bonne évaluation du modèle. En effet, le but d'un modèle de détection n'est pas de classifier correctement les données d'apprentissage, mais d'être capable de généraliser, c'est-à-dire de classifier correctement des données non utilisées au cours de sa phase d'apprentissage. Cependant, analyser les performances du modèle sur les données d'apprentissage peut permettre de détecter le problème de *sous-apprentissage*.

On dit qu'il y a *sous-apprentissage* quand le modèle de détection n'a presque rien appris à partir des données d'apprentissage. Dans cette



(a) Modèle linéaire insuffisant (b) Modèle quadratique suffisant

Fig. 6. Illustration du sous-apprentissage. Les données avec une croix correspondent aux erreurs de classification.

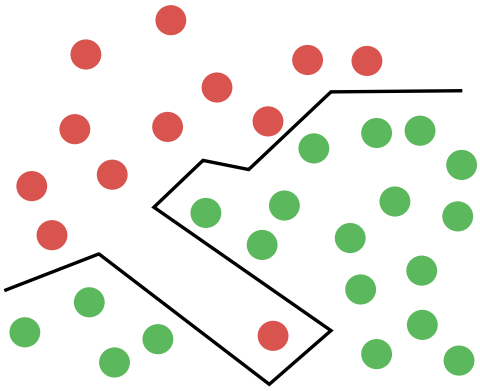
situation, le modèle de détection se comporte presque comme un générateur aléatoire. Pour diagnostiquer le sous-apprentissage, il suffit de tracer la courbe ROC obtenue sur les données d'apprentissage et de vérifier qu'elle n'est pas trop proche de celle d'un générateur aléatoire (cf. courbe aléatoire de la figure 5). Deux options s'offrent à l'expert pour résoudre le sous-apprentissage : ajouter des attributs discriminants ou utiliser un type de modèle de classification plus complexe.

Lorsqu'un modèle n'arrive pas à discriminer les instances malveillantes des bénignes sur les données d'apprentissage, c'est souvent que l'expert n'a pas donné de bons attributs en entrée. Il est donc nécessaire qu'il reprenne la phase d'extraction d'attributs pour ajouter des caractéristiques plus discriminantes. Parfois, l'expert a fourni des attributs discriminants, mais le type de modèle de classification choisi n'est pas suffisamment complexe pour discriminer les instances malveillantes des bénignes. La figure 6(a) montre un jeu de données en deux dimensions où un modèle linéaire n'est pas suffisamment complexe pour séparer correctement les instances malveillantes des bénignes, alors qu'un modèle quadratique, légèrement plus complexe, est parfaitement adapté (cf. figure 6(b)).

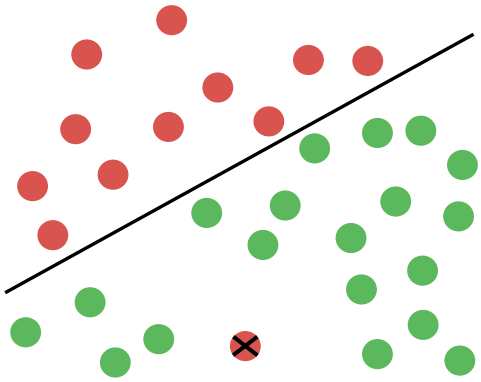
Utiliser un modèle de détection extrêmement complexe n'est cependant pas une solution, car dans ce cas on risque le *sur-apprentissage*. On dit qu'il y a *sur-apprentissage* quand le modèle de classification prédit parfaitement le label des données d'apprentissage, mais qu'il est incapable de prédire correctement le label de nouvelles données. Le problème de sur-apprentissage peut être diagnostiqué en analysant les performances du modèle sur un jeu de validation indépendant et est donc présenté dans la section suivante.

5.2 Validation sur un jeu de données indépendant

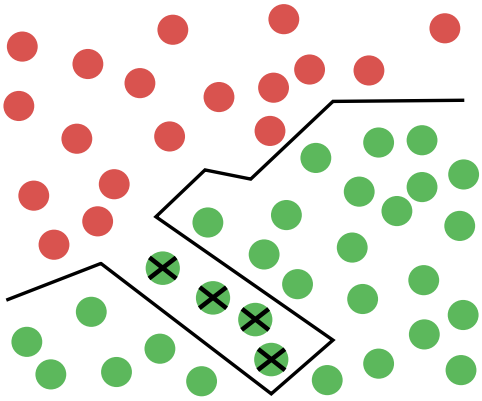
Une fois que le modèle a une performance convenable sur les données d'apprentissage, on peut passer à la deuxième phase de validation. Le but cette fois est de vérifier que le modèle de détection est capable de généraliser, c'est-à-dire qu'il a de bonnes performances de détection sur des données de validation indépendantes non utilisées au cours de la phase d'apprentissage.



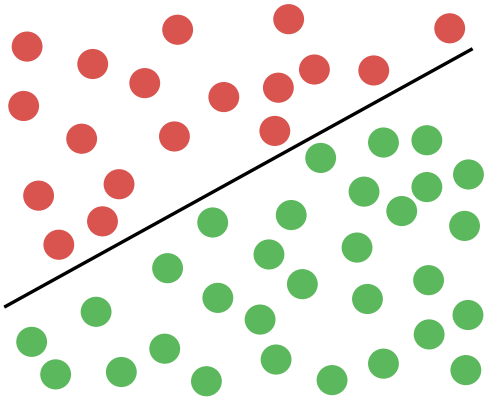
(a) *Modèle complexe :* pas d'erreur d'apprentissage



(b) *Modèle simple :* une erreur d'apprentissage sur une donnée aberrante



(c) *Modèle complexe :* de nombreuses erreurs sur les données de validation



(d) *Modèle simple :* pas d'erreur sur les données de validation

Fig. 7. Illustration du sur-apprentissage. La ligne du haut représente les données d'apprentissage, et la ligne du bas les données de validation. Les données avec une croix correspondent aux erreurs de classification.

Si les performances sur les données de validation ne sont pas concluantes, le modèle souffre certainement de *sur-apprentissage*. Ce problème est représenté par la figure 7. Lorsque le modèle de détection est

trop complexe (cf. figures 7(a) et 7(c)), il permet de prédire parfaitement le label de toutes les données d'apprentissage (cf. figure 7(a)), mais il commet de nombreuses erreurs de prédiction sur le jeu de validation (cf. figure 7(c)). En effet, le modèle complexe est parfaitement adapté aux données d'apprentissage, mais il a de faibles capacités de généralisation. En revanche, un modèle plus simple (cf. figures 7(b) et 7(d)) est apte à éviter la donnée aberrante (*outlier* en anglais) pour obtenir un modèle généralisant beaucoup mieux sur des données de validation.

Un modèle de détection peut faire des erreurs de prédiction sur les données d'apprentissage, le plus important est qu'il soit capable de généraliser sur de nouvelles données. Les données d'apprentissage peuvent contenir des données aberrantes ou des erreurs de labélisation, que l'algorithme d'apprentissage ne doit pas prendre en compte lors de la construction du modèle pour avoir de meilleures capacités de généralisation.

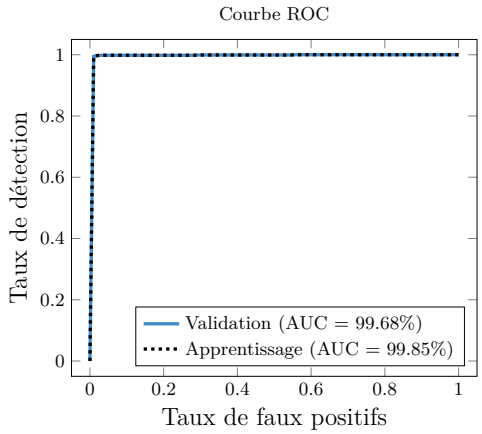
Finalement, le modèle de détection doit ainsi être ni trop simple pour éviter le sous-apprentissage, ni trop complexe pour éviter le sur-apprentissage. Tout d'abord, il faut vérifier que le modèle utilisé est suffisamment complexe pour discriminer les instances malveillantes des bénignes sur les données d'apprentissage. Ceci revient à vérifier que la courbe ROC obtenue sur les données d'apprentissage s'éloigne de celle d'un générateur aléatoire, mais elle n'a pas besoin de ressembler à celle d'un classifieur optimal. Ensuite, il faut vérifier que le modèle a de bonnes capacités de généralisation. Ceci revient à vérifier que le modèle a une bonne courbe ROC sur des données de validation non utilisées au cours de la phase d'apprentissage.

5.3 Diagnostic des biais d'apprentissage

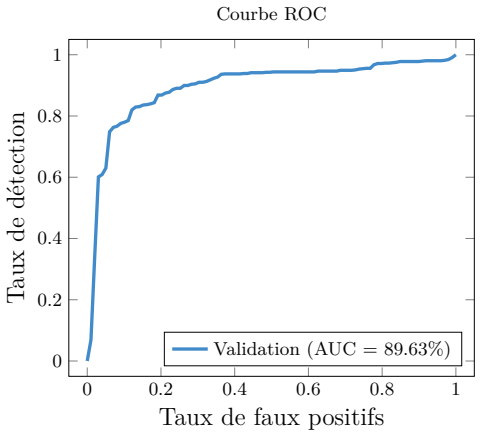
Une fois que les performances du modèle de détection sont satisfaisantes aussi bien sur le jeu d'apprentissage, que sur le jeu de validation, on peut passer à la phase de test en production. Cependant, les données d'apprentissage peuvent ne pas être représentatives des données de production. On se trouve alors dans une situation de *biais d'apprentissage* : le classifieur ne sera pas aussi efficace qu'attendu. Les biais d'apprentissage peuvent être détectés au moment de la mise en production en analysant les alertes générées, mais des techniques peuvent permettre de les anticiper.

Biais d'apprentissage Tout d'abord, nous montrons un exemple de biais d'apprentissage avec le cas de la détection de fichiers PDF malveillants. Nous apprenons le modèle de détection sur le jeu d'apprentissage

Contagio_90%, et nous le testons sur le jeu de validation Contagio_10%. La figure 8(a) montre les résultats des deux phases de validation présentées dans les sections 5.1 et 5.2. Le modèle est très efficace pour classifier les PDF de Contagio : il a de très bonnes performances aussi bien sur les données d'apprentissage que sur le jeu de validation.



(a) Apprentissage sur Contagio_90% et validation sur Contagio_10%



(b) Apprentissage sur Contagio et validation sur webPdf

Fig. 8. Courbes ROC permettant la détection d'un biais d'apprentissage

Ces bons résultats sont encourageants pour commencer l'étape de mise en production du modèle de détection. Pour simuler la mise en production, nous apprenons un modèle avec l'intégralité des données de Contagio, et nous l'appliquons au jeu de données webPdf. La figure 8(b) montre les performances du modèle sur webPdf. On remarque que le classifieur n'est pas aussi efficace sur webPdf que sur Contagio : l'AUC sur webPdf est seulement de 89.63% contre 99.68% sur Contagio.

Le modèle que nous avons appris n'est en fait pas capable de détecter des PDF malveillants dans le cas général, mais seulement de parfaitement distinguer les PDF malveillants des PDF bénins de Contagio. Cette simulation de mise en production nous permet de présenter un problème qui peut apparaître avec les méthodes d'apprentissage automatique : le *biais d'apprentissage*.

La simulation de mise en production avec webPdf nous a permis de facilement détecter un biais d'apprentissage parce que le jeu de données est entièrement labélisé. Or, les labels des données issues de la production sont inconnus en pratique ce qui rend la validation très coûteuse. En effet, valider un modèle en production nécessite qu'un expert vérifie

l'intégralité des prédictions manuellement afin d'estimer non seulement le taux de fausses alertes, mais aussi le taux de détection. Nous allons maintenant présenter une méthode permettant de diagnostiquer les biais d'apprentissage avant de faire des tests coûteux en production.

Diagnostic L'étude des indicateurs de performance du modèle sur les données d'apprentissage et de validation n'est pas suffisante pour détecter d'éventuels biais d'apprentissage en amont de la mise en production : une analyse approfondie du fonctionnement du modèle est nécessaire. Comme expliqué dans la section 4.3, le modèle de classification que nous avons choisi est interprétable. Nous allons utiliser cette interprétabilité pour diagnostiquer d'éventuels biais d'apprentissage. Cette technique d'analyse ne serait pas possible avec un réseau de neurones par exemple, car la complexité du modèle rend l'interprétation impossible.

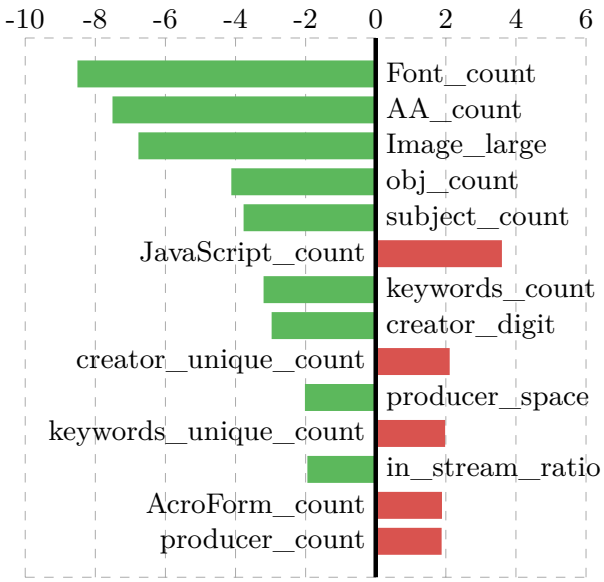


Fig. 9. Coefficients les plus importants du modèle de détection

La figure 9 présente les coefficients des attributs ayant le plus d'impact dans la prise de décision du modèle de détection appris sur le jeu de données Contagio. On trouve, parmi les attributs les plus importants de ce modèle : *Font_count*, *AA_count*, *Image_large* et *obj_count*. Ces attributs, qui représentent le nombre de polices embarquées, le nombre d'actions automatiques, le nombre de grandes images et le nombre total d'objets sont étroitement liés avec la complexité du fichier PDF. En effet, un fichier PDF complet contiendra un grand nombre d'objets, d'images et

Moyennes	Contagio	webPdf
Font_count	**6.14**	**29.18**
AA_count	**3.46**	**22.6**
Image_large	**0.12**	**2.86**
obj_count	**46.52**	**239.63**
subject_count	0.20	0.17
JavaScript_count	0.82	1.05
keywords_count	0.31	0.20
creator_digit	1.84	9.94
creator_unique_count	0.51	0.89
producer_space	1.65	2.26
keywords_unique_count	0.18	0.13
in_stream_ratio	0.82	0.83
AcroForm_count	0.30	0.21
producer_count	0.85	1.75

Tableau 1. Moyennes des attributs de la figure 9

	Contagio	webPdf
Taille moyenne des fichiers	56ko	957ko
Nombre moyen de pages des fichiers	3.4	22.5

Tableau 2. Caractéristiques des deux jeux de données

de polices embarquées. On remarque que les coefficients de ces attributs sont négatifs, et que le modèle aura donc tendance à considérer qu'un fichier complexe sera sain. À l'inverse, la présence de JavaScript sera un indice de malveillance, ce qui explique le coefficient positif de l'attribut *JavaScript_count*. Une première interprétation de ces résultats consiste à remarquer que les PDF malveillants qui composent Contagio sont souvent petits et simples, alors que les PDF sains sont plus complets. En effet, la plupart de ces PDF malveillants comportent uniquement une charge JavaScript malveillante qui sera exécutée à l'ouverture. Ces fichiers très simples, constitués d'une seule page sans images ni polices, sont très différents des PDF sains qui eux sont plus complexes avec un grand nombre d'objets, d'images et de polices embarquées.

Le tableau 1 présente les statistiques descriptives de ces attributs sur le jeu d'apprentissage et sur les données de production. Cette table fait apparaître d'importantes disparités entre les deux jeux de données. En particulier, le nombre moyen d'objets de type *Police* (attribut *Font_count*), et le nombre moyen de grandes images (attribut *Image_large*) diffèrent grandement. Ces écarts importants permettent d'expliquer pourquoi un modèle appris sur Contagio peut être inefficace sur webPdf.

Avec cette expérience, nous avons montré que le jeu d'apprentissage doit être adapté aux données du contexte de déploiement du système de détection. Il doit être suffisamment représentatif des données qui seront traitées en production pour permettre une classification efficace. Cette expérience a aussi mis en avant qu'il faut faire particulièrement attention aux jeux de données publics. Contagio est très pratique pour comparer des méthodes de détection dans le monde académique, mais il ne permet pas d'apprendre un modèle de détection efficace en production, car ses fichiers sont trop différents de ceux analysés par un système de détection en production.

Cette notion de proximité entre les données d'apprentissage et le contexte de détection est assez complexe à évaluer. Il est très difficile d'estimer l'efficacité de détection qu'aura un modèle appris sur un jeu de données sans réaliser de tests. Une comparaison entre les caractéristiques des données peut donner une première idée du biais d'apprentissage dont souffrira notre modèle. Cette comparaison est présentée dans le tableau 2, et les différences importantes dans la taille des fichiers ou dans le nombre de pages permettent de se rendre compte de l'écart entre Contagio et webPdf. Cet écart, que l'on aurait pu observer avant même d'apprendre un modèle de détection, nous montre que les jeux sont très différents et qu'il sera par conséquent difficile d'obtenir de bonnes performances.

Instinctivement, on comprend assez bien qu'il est plus facile de reconnaître un félin si l'on a déjà vu un tigre, une panthère et un lion que si l'on ne connaît que les chatons.

Réduction des biais d'apprentissage Nous proposons deux solutions pour réduire l'impact des biais d'apprentissage lorsque les données d'apprentissage disponibles ne sont pas adaptées aux données en production.

Dans certains cas, il est possible d'ajouter une phase intermédiaire d'adaptation du modèle en production. Grâce à l'analyse des alertes, en réinjectant des faux positifs et des vrais positifs dans les données d'apprentissage, il est possible d'améliorer le modèle en corrigeant une partie du biais d'apprentissage. Cette méthode a cependant un défaut : certains faux négatifs causés par le biais d'apprentissage peuvent ne jamais être détectés.

Il est aussi possible d'utiliser un échantillon des données en productions pour améliorer l'apprentissage. Cette méthode nécessite un expert qui va labéliser un échantillon représentatif des données en production. Cette méthode peut être très coûteuse pour générer d'importants jeux d'apprentissage, une méthode interactive pouvant assister l'expert est présentée dans la section 6.

Le modèle de détection doit évoluer au cours du temps pour suivre la menace et prendre en compte les retours des experts en sécurité. Les modèles linéaires permettent de mettre à jour le modèle facilement avec de nouvelles données de façon incrémentale. Ce processus permet d'affiner progressivement le modèle en rajoutant des données d'apprentissage issues directement du contexte de détection.

En conclusion, nous tenons à préciser qu'appliquer une fois les étapes de construction et de validation du modèle, présentées dans les sections 4 et 5, n'est en général pas suffisant. En pratique, la mise en place d'un modèle est un processus itératif où la phase de validation permet d'améliorer l'apprentissage du modèle.

À chaque itération, l'expert réalise la phase d'apprentissage d'un modèle de détection et procède à sa validation. En fonction des résultats, l'expert peut soit considérer que le modèle de détection est satisfaisant, soit faire une nouvelle itération en modifiant les données d'apprentissage, les attributs ou le type de modèle. En général, les premières itérations consistent à ajouter de nouveaux attributs. Ensuite, il est très simple de tester plusieurs types de modèles avec les bibliothèques dédiées à l'apprentissage automatique une fois que les données d'apprentissage sont construites. En revanche, la modification des données d'apprentissage n'est

en général considérée qu'en dernier ressort, car elle implique souvent qu'un expert annote des données manuellement.

La mise en place d'un modèle de détection est un processus itératif qui doit se dérouler hors ligne pour éviter les surcoûts liés à la production. En effet, valider un modèle en production nécessite que l'opérateur vérifie à la main certaines des alertes levées, et les faux négatifs sont en général difficiles à estimer. Une fois que les phases de validation ont été effectuées avec succès, il est envisageable de faire des tests en production. Dans la section suivante, nous présentons l'outil SecuML que nous avons développé pour faciliter la mise en place de modèles de détection supervisés.

6 SecuML

SecuML[8] est un outil libre qui permet d'appliquer diverses méthodes d'apprentissage automatique à des problèmes de détection d'intrusion. Il permet notamment d'apprendre des modèles de détection supervisés, et de les valider facilement avec les bonnes pratiques présentées dans la section précédente. SecuML s'appuie sur scikit-learn [15] pour l'apprentissage des modèles, et propose une interface web pour visualiser leurs performances et diagnostiquer d'éventuels biais d'apprentissage.

Les services en ligne tels que Google Cloud ML, Microsoft Azure, ou Amazon Machine offrent des solutions de visualisation des performances des modèles, mais ces services demandent de mettre les données dans le *cloud* ce qui est incompatible avec les systèmes de détection en général. Des publications académiques [1, 10, 14, 22] présentent aussi des interfaces pour faciliter l'utilisation de l'apprentissage automatique par des non-experts. Cependant, ces papiers présentent uniquement des captures d'écran, et l'absence d'interfaces publiques entrave fortement leur utilisation en pratique [4, 13, 24].

SecuML a été conçu spécialement pour les problèmes de détection d'intrusion pour favoriser l'intégration de l'apprentissage automatique dans les systèmes de détection. SecuML n'a pas vocation à être utilisé en production, mais à faciliter la mise en place et la validation de modèles de détection supervisés.

Construction du modèle de détection SecuML est un outil générique pouvant construire des modèles de détection supervisés dont le type est choisi par l'expert.

[8] `https://github.com/ANSSI-FR/SecuML`

Afin d'apprendre un modèle de détection, l'expert doit fournir des données d'apprentissage : les attributs numériques décrivant chacune des instances ainsi que leur label. La phase d'extraction des attributs est spécifique à chaque problème de détection et doit donc être effectuée par l'expert en dehors de SecuML. En revanche, dans le cas où l'expert n'aurait pas de données labélisées à sa disposition, il peut utiliser SecuML pour faciliter la phase d'annotation. En effet, SecuML intègre un composant appelé ILAB (pour Interactive LABeling, labélisation interactive en français) qui permet d'obtenir un jeu de données labélisées représentatif d'un réseau supervisé *in situ* afin d'éviter tout biais d'apprentissage. Le principe général est de récupérer un jeu de données non labélisées à partir du réseau supervisé, et un algorithme d'apprentissage actif [17] va demander à l'expert d'annoter, avec un label bénin ou malveillant, certaines instances sélectionnées de sorte à améliorer le classifieur.

Fig. 10. Interface de validation d'un modèle de détection de SecuML

Validation du modèle de détection SecuML facilite la validation d'un modèle de détection avec les bonnes pratiques présentées dans la section 5. L'interface affiche des indicateurs de performance du modèle de détection sur ses données d'apprentissage et sur des données de validation : le taux de faux positifs et le taux de détection pour un seuil de détection choisi par l'expert via un curseur, ainsi que la courbe ROC avec l'AUC correspondante (cf. figure 10).

L'interface permet également de faire une analyse plus fine des prédictions du modèle. En effet, elle affiche la matrice de confusion, les instances mal classifiées par le modèle de détection ainsi que l'histogramme des probabilités de malveillance prédites pour les instances bénignes et malveillantes (cf. partie droite de la figure 10). L'étude des erreurs de prédiction peut aider l'expert à trouver de nouveaux attributs discriminants, ou à repérer d'éventuelles erreurs de labélisation. L'histogramme des probabilités prédites permet notamment à l'expert d'analyser les instances mal classifiées par le modèle avec un fort taux de confiance ou ceux proches de la frontière de décision (probabilité de malveillance proche de 50%) pour lesquelles le modèle de détection a beaucoup de difficulté à prendre une décision. Analyser ces instances peut aussi aider l'expert à trouver de nouveaux attributs pertinents.

Enfin, l'interface permet de réaliser quelques étapes nécessaires avant la mise en production. Les coefficients associés par le modèle à chaque attribut sont visualisés, comme dans la figure 9, afin de pouvoir détecter d'éventuels biais d'apprentissage. Par ailleurs, l'interface permet d'accéder à un mode supervision affichant les alertes générées par le modèle sur les données de validation pour un seuil de détection donné. Enfin, l'interface permet à l'administrateur du système de détection de fixer le seuil de détection en fonction du taux de détection souhaité ou du taux de fausses alertes toléré.

7 Conclusion

Dans cet article, nous avons décrit une méthodologie pour résoudre l'aspect « boîte noire » souvent reproché aux méthodes d'apprentissage automatique, et nous l'avons illustrée avec un cas d'étude sur la détection de fichiers PDF malveillants. Grâce à cette méthodologie, il devient possible d'adapter l'apprentissage automatique aux réticences des experts en sécurité, et aux contraintes des systèmes de détection d'intrusion.

L'article met en particulier l'accent sur l'importance de la rigueur de la méthode de validation du modèle avant sa mise en production. En effet, des erreurs lors de la phase de validation peuvent surestimer l'efficacité d'un modèle, et donc conduire à de mauvaises surprises lors de la mise en production. La méthodologie de validation que nous proposons permet d'anticiper des problèmes inhérents à l'apprentissage automatique, et donc d'éviter des surcoûts liés aux tests en production. L'outil SecuML, disponible sur le GitHub de l'ANSSI, permet de mettre en œuvre facilement nos recommandations.

Par ailleurs, les modèles reposants sur de l'apprentissage automatique ne fournissent pas seulement une réponse binaire, mais une probabilité de malveillance. L'administrateur du système de détection peut ainsi faire un compromis entre taux de détection et taux de faux positifs par l'intermédiaire du seuil de détection. Cette probabilité de malveillance offre aussi la possibilité de donner une priorité aux alertes devant être traitées par l'opérateur.

Enfin, il est crucial que le modèle de détection soit interprétable pour être accepté par les experts en sécurité. Un modèle interprétable facilite l'exploitation des alertes générées par l'opérateur du système de détection, et l'administrateur peut comprendre son fonctionnement avant sa mise en production. Un modèle non interprétable empêcherait les experts en sécurité d'utiliser leurs connaissances métier pour vérifier sa cohérence avec la cible de détection.

L'apprentissage automatique peut donc être intégré aux systèmes de détection, en complément des signatures, pour améliorer la détection des nouvelles menaces. La prise en compte des contraintes opérationnelles lors du choix du modèle, et la rigueur de la validation permettront d'éviter les principaux écueils lors d'un déploiement opérationnel.

Références

1. Saleema Amershi, Max Chickering, Steven M Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. Modeltracker : Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 337–346. ACM, 2015.
2. Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots : detecting the rise of DGA-based malware. In *USENIX Security*, pages 491–506, 2012.
3. Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure : detecting botnet command and control servers through large-scale netflow analysis. In *ACSAC*, pages 129–138, 2012.
4. Dong Chen, Rachel KE Bellamy, Peter K Malkin, and Thomas Erickson. Diagnostic visualization for non-expert machine learning practitioners : A design study. In *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on*, pages 87–95. IEEE, 2016.
5. Mihai Christodorescu and Somesh Jha. Testing malware detectors. *ACM SIGSOFT Software Engineering Notes*, 29(4) :34–44, 2004.
6. Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0r : Detection of malicious pdf-embedded javascript code through discriminant analysis of api references. In *AISEC*, pages 47–57, 2014.
7. Dorothy E Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2) :222–232, 1987.

8. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.
9. James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1) :29–36, 1982.
10. Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1343–1352. ACM, 2010.
11. Khaled N Khasawneh, Meltem Ozsoy, Caleb Donovick, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Ensemble Learning for Low-Level Hardware-Supported Malware Detection. In *RAID*, pages 3–25. 2015.
12. Pavel Laskov and Nedim Šrndić. Static detection of malicious JavaScript-bearing PDF documents. In *ACSAC*, pages 373–382, 2011.
13. Oisin Mac Aodha, Vassilios Stathopoulos, Gabriel J Brostow, Michael Terry, Mark Girolami, and Kate E Jones. Putting the Scientist in the Loop–Accelerating Scientific Progress with Interactive Machine Learning. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 9–17. IEEE, 2014.
14. Thorsten May, Andreas Bannach, James Davey, Tobias Ruppert, and Jörn Kohlhammer. Guiding feature subset selection with an interactive visualization. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 111–120. IEEE, 2011.
15. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine Learning in Python. *JMLR*, 12 :2825–2830, 2011.
16. Konrad Rieck. Computer Security and Machine Learning : Worst Enemies or Best Friends ? In *SysSec*, pages 107–110, 2011.
17. Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1) :1–114, 2012.
18. Yun Shen and Olivier Thonnard. MR-TRIAGE : Scalable multi-criteria clustering for big data security intelligence applications. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 627–635. IEEE, 2014.
19. Charles Smutz and Angelos Stavrou. Malicious PDF detection using metadata and structural features. George Mason University.
20. Charles Smutz and Angelos Stavrou. Malicious PDF detection using metadata and structural features. In *ACSAC*, pages 239–248, 2012.
21. Nedim Šrndic and Pavel Laskov. Detection of malicious pdf files based on hierarchical document structure. In *NDSS*, 2013.
22. Justin Talbot, Bongshin Lee, Ashish Kapoor, and Desney S Tan. EnsembleMatrix : interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1283–1292. ACM, 2009.
23. Kalyan Veeramachaneni and Ignacio Arnaldo. AI2 : Training a big data machine to defend. In *DataSec*, pages 49–54, 2016.
24. Kiri L Wagstaff. Machine Learning that Matters. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 529–536, 2012.