
SR2I208 - Projet de filière

Compte rendu

*Détection et classification de malwares en utilisant
différents modèles d'intelligences artificielle*

<https://github.com/NathSimon/SR2I208>

Nathanaël Simon

Hamza Zarfaoui

Eddy Raingeaud

`nathanael.simon@telecom-paris.fr`

`hamza.zarfaoui@telecom-paris.fr`

`eddy.raingeaud@telecom-paris.fr`

TÉLÉCOM PARIS, INSTITUT POLYTECHNIQUE DE PARIS

June 28, 2023

Abstract

Dans le cadre de la filière Sécurité des Réseaux et des Infrastructures Informatiques (SR2I), les étudiants ont l'opportunité en deuxième année de mettre en pratique les compétences acquises au cours de la formation dans un projet de filière. Ce dernier permet également de développer et croiser de nouvelles compétences, non abordées directement en SR2I, afin de se former dans des domaines étendus.

Le but de ce document est de rendre compte du projet mené par Hamza Zarfaoui, Nathanaël Simon et Eddy Raingeaud, intitulé "Détection et classification de malwares en utilisant différents modèles d'intelligences artificielle.

Il sera ici question d'expliquer le projet et ses objectifs, les méthodes utilisées, puis de discuter les résultats obtenus en cherchant des pistes d'améliorations. Nous verrons ainsi l'implémentation de différents modèles populaires d'intelligences artificielle appliqués à la cybersécurité.

Contents

1	Présentation	4
2	Datasets	5
2.1	Detection	5
2.2	Classification	6
2.3	Création des sous-sets	8
3	Modèles utilisés	8
3.1	Machine learning	8
3.1.1	Support Vector Machine	8
3.1.2	AdaBoost	9
3.1.3	Extreme Gradient Boost	10
3.1.4	Multi-Layer Perceptron Classifier	10
3.1.5	Random Forest	11
3.2	Deep Learning	12
3.2.1	Sequential Model	12
4	Evaluation	12
4.1	Difference de l'évaluation entre classification multi-label et multi-class . .	12
4.2	Accuracy	13
4.3	Recall	14
4.4	Precision	14
4.5	F1-Score	15
4.6	Macro-average F1-Score	15
4.7	Weighted-average F1-Score	15
4.8	Courbe ROC et score AUC	16
5	Resultats	17
5.1	Détection	18
5.1.1	Classification Reports	18
5.1.2	Courbes ROC et scores AUC	19

5.2	Classification	26
5.2.1	Classification Reports	26
5.2.2	Scores AUC et courbes ROC	28
6	Discussion	37
6.1	Détection	37
6.2	Classification	38
6.2.1	Analyse des données	39
7	Annexes	42
	References	47

1 Présentation

L'intelligence artificielle connaît depuis les 15 dernières années un essor exceptionnel, poussé par des progrès théoriques et techniques, qui permettent des avancées et applications de plus en plus importantes et diversifiées. La cybersécurité est un domaine particulièrement impacté par ces progrès, aussi bien sur le côté offensif que défensif.

Ces dernières années, de nombreuses entreprises ont commercialisés des outils de détection d'intrusions et de classification de malware reposant sur des techniques d'intelligence artificielle, utilisant des modèles de Machine Learning que de Deep Learning.

Alors que le monde industriel connaît un changement profond, nous avons souhaité, au cours de ce projet, nous former sur ces différentes techniques. Etant nous trois des élèves en alternance, sous-statut FISA, nous avons une coloration très forte en cybersécurité, mais assez peu de couverture des sujets sur la science des données. Nous souhaitons alors, par l'exploration des thématiques couvertes par ce sujet, ouvrir notre formation sur ce domaine.

Le but du projet est de réaliser une détection ainsi qu'une classification de 8 types de malwares différents : Trojan, Adware, Dropper, Downloader, Worms, Backdoor, Spyware et Virus.

Nous ne tenterons pas de réaliser une classification à partir du code source de ces derniers, ou bien de leur hash mais à partir des appels API système de Windows lors de son exécution, alors infecté par un de ces malwares. Ceci permet d'être agnostique du malware et des techniques de détection par comparaisons du hash, typiquement utilisés par les anti-virus du marché.

Nous explorons alors une nouvelle méthode pour réaliser de la détection et classification de malware, en aval de l'exécution de ce dernier dans le système. Cela présente un inconvénient, qui est la nécessité d'exécution du programme malveillant, et nous empêche d'être préventif. Cependant, cela peut nous permettre de détecter tout malware rentrant dans une des huit catégories citées précédemment, même si celui-ci n'est pas référencé par les anti-virus.

Nous avons traité le problème de la détection et de la classification de manière séparée. Sur un premier dataset [1] [2], nous avons des API calls sains et malveillants, dans lequel nous réaliserons donc de la détection, et dans un second [3] nous avons des API calls contenant un des huit malwares possibles, dans lequel nous réaliserons de la classification.

Il est possible de fusionner ces deux datasets pour réaliser ces deux tâches d'un coup. On se ramènerait alors à un problème de classification entre les 8 malwares et une nouvelle classe saine. Cependant, comme nous le verrons par la suite, cela présente des résultats et efficacités différentes.

Les productions de ce projet sont référencées dans les annexes. Cela contient, en plus de ce document, les deux notebooks python utilisés et une application permettant la visualisation des résultats obtenus ainsi que la détection en temps réel.

Il est important de noter que ce projet est à but exploratoire pour nous, et ne vise pas à rendre le projet le plus performant possible, mais plutôt de comprendre les techniques que nous mettons en place, et de pouvoir interpréter les résultats obtenus, tout en analysant les pistes d'améliorations possibles.

Au cours de ce document, nous présenterons donc les datasets que nous avons utilisés et le traitement de données que nous avons appliqué. Nous discuterons ensuite des modèles implémentés, de leur méthode de fonctionnement et pertinence pour la tâche demandée. Ensuite nous expliquerons les méthodes d'évaluation mise en place pour comparer les différents modèles. Enfin, nous discuterons des résultats obtenus, et les pistes d'améliorations, en réalisant une analyse des données d'entrée.

2 Datasets

Nous avons utilisés dans ce projet deux datasets aux objectifs différents. Le premier est un dataset présentant des api calls sains et d'autres contenant des malwares [1] [2]. On se retrouve donc dans un problème de classification binaire, nous permettant de réaliser une détection de malware.

Dans un second dataset, nous possédons les suites d'api calls, tagué avec le malware correspondant [3]. Il n'y a pas de calls sains. Il nous est alors possible de réaliser de la classification, et non plus de la détection. Le problème est alors un problème de classification multi-class.

2.1 Detection

Le premier dataset [1] [2] pour la detection est déjà pré-traité pour une utilisation adaptée à la science des données. Il contient des entrées sous cette forme :

	0	1	2	3	4	5	6	7	8	9	...	92	93	94	95	96	97	98	99	100	101
0	hash	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	...	t_91	t_92	t_93	t_94	t_95	t_96	t_97	t_98	t_99	malware
1	071e8c3f8922e186e57548cd4c703a5d	112	274	158	215	274	158	215	298	76	...	71	297	135	171	215	35	208	56	71	1
2	33f8e6d08a6aae939f25a8e0d63dd523	82	208	187	208	172	117	172	117	172	...	81	240	117	71	297	135	171	215	35	1
3	b68abd064e975e1c6d5f25e748663076	16	110	240	117	240	117	240	117	240	...	65	112	123	65	112	123	65	113	112	1
4	72049be7bd30ea61297ea624ae198067	82	208	187	208	172	117	172	117	172	...	208	302	208	302	187	208	302	228	302	1

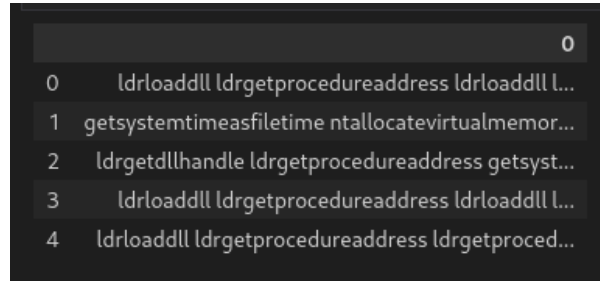
Figure 1: Dataset de détection

Chaque sample se compose des features suivantes :

- hash : Le hash MD5 du sample
- t1-t100 : Les 100 premiers appels api sans répétitions vectorisés
- Malware : étiquette du sample. 1 signifie la présence d'un malware, 0 un sample sain

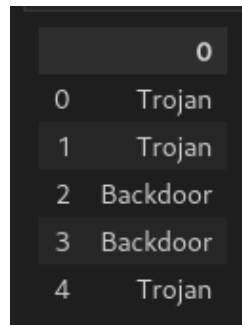
2.2 Classification

Le second dataset [3] possède une suite d'api calls sous la forme de strings, ainsi qu'un fichier contenant tous les labels pour chaque sample correspondant à son index.



	0
0	ldrloaddll ldrgetprocedureaddress ldrloaddll l...
1	getsystemtimeasfiletime ntallocatevirtualmemor...
2	ldrgetdllhandle ldrgetprocedureaddress getsyst...
3	ldrloaddll ldrgetprocedureaddress ldrloaddll l...
4	ldrloaddll ldrgetprocedureaddress ldrgetproced...

Figure 2: Dataset de classifications



	0
0	Trojan
1	Trojan
2	Backdoor
3	Backdoor
4	Trojan

Figure 3: Labels associés

Ce dataset sous cette forme sera difficilement exploitable. Dans un premier temps, nous allons vectoriser les api calls. L'idée est d'associer à chaque call un entier unique, qui sera facilement traitable par un modèle. Pour se faire, nous utilisons la fonction `CountVectorizer` de `skicit-learn`. On peut voir dans la figure 4 ci-dessous le vocabulaire retenu et les entiers associés, et dans la figure 5 le dataset transformé avec les valeurs précédentes.

Cependant, ce dataset ne contient pas de features, et il ne sera donc pas possible de réaliser de la feature selection.

```

output exceeds the FILE limit: open e
{'ldrloaddll': 132,
 'ldrgetprocedureaddress': 131,
 'regopenkeyexa': 221,
 'ntopenkey': 169,
 'ntqueryvaluekey': 181,
 'ntclose': 150,
 'ntqueryattributesfile': 176,
 'loadstringa': 136,
 'ntallocatevirtualmemory': 149,
 'ldrgetdllhandle': 130,
 'ldrunloaddll': 133,
 'findfirstfileexw': 52,
 'copyfilea': 13,
 'regcreatekeyexa': 209,
 'regsetvalueexa': 227,
 'regclosekey': 208,
 'createprocessinternalw': 21,
 'ntfreevirtualmemory': 164,
 'ntterminateprocess': 190,
 'getsystemtimeasfiletime': 92,
 'setunhandledexceptionfilter': 252,
 'ntcreatemutant': 153,
 'getsysteminfo': 90,
 'getsystemdirectoryw': 89,
 '__exception__': 1,
 ...
 'rtlcreateuserthread': 234,
 'setinformationjobobject': 249,
 'cryptprotectmemory': 37,
 'cryptunprotectmemory': 39,
 'findfirstfileexa': 51}

```

Figure 4: Vocabulaire de la vectorisation

```

array([[0, 0, 0, ..., 0, 0, 0],
       [0, 3, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])

```

Figure 5: Résultat de la vectorisation

2.3 Création des sous-sets

Afin de pouvoir entraîner et tester nos différents modèles, il est nécessaire de séparer le dataset en 2 sous-sets distincts. Un premier servira à l'entraînement, tandis que l'autre à l'évaluation. Ainsi, les modèles ne seront pas testés sur des valeurs sur lesquelles ils se sont entraînés.

Le ratio de ces deux sous-sets peut varier dans les implémentations, mais il est généralement compris entre 70/30 ou 80/20. Dans notre cas nous avons choisi une répartition 80/20, permettant un nombre important de données d'entraînement, tout en évitant les problèmes d'overfitting. [4]

3 Modèles utilisés

Pour la réalisation de ce projet, nous avons choisi 6 modèles différents, 5 de machine learning et 1 de deep learning. Nous utiliserons les 6 à la fois pour la classification binaire et multi-class, pour pouvoir comparer les algorithmes entre eux sur un même problème, et également sur des problèmes différents.

Nous avons choisi les algorithmes populaires pour faire de la classification, afin de faire un tour des modèles les plus utilisés actuellement.

Enfin, nous avons fait le choix de nous concentrer principalement sur les algorithmes de Machine Learning, le fonctionnement des algorithmes de deep learning étant plus abstrait.

3.1 Machine learning

3.1.1 Support Vector Machine

Le modèle Support Vector Machine (SVM) est une méthode d'apprentissage supervisé utilisée pour la classification et la régression. L'idée principale derrière son fonctionnement est de trouver un hyperplan optimal qui sépare les différentes classes de données. L'objectif est de maximiser la marge entre les points de données les plus proches de chaque classe, appelés vecteurs de support, et l'hyperplan de décision. Cela permet de garantir une bonne généralisation et une capacité à traiter des données non linéaires, en utilisant la technique du kerneling.

Le concept de kernelling, également connu sous le nom de fonction de noyau, est une composante essentielle des SVM et d'autres méthodes d'apprentissage automatique. Il permet de transformer les données d'entrée dans un espace de dimension supérieure, où la séparation des classes devient plus facile.

Les SVM utilisent différents types de kernels pour transformer les données d'entrée dans un espace de dimension supérieure, où la séparation des classes peut être plus facilement réalisée. Les kernels les plus couramment utilisés sont le linéaire, le polynomial

et le gaussien (ou RBF). Le kernel linéaire est utilisé lorsque les données sont linéairement séparables, tandis que les kernels polynomial et gaussien sont plus adaptés aux données non linéaires.

Les avantages des SVM résident dans leur capacité à traiter efficacement des ensembles de données de grande dimension et dans leur résistance aux problèmes de sur-ajustement. Ils peuvent également fonctionner avec des données non linéaires grâce à l'utilisation de kernels. De plus, les SVM peuvent gérer des ensembles de données contenant des valeurs manquantes.

Cependant, le choix du kernel approprié peut être délicat, et il peut y avoir des compromis entre la complexité du modèle et sa capacité de généralisation. Une mauvaise sélection de kernel peut entraîner un sur-ajustement ou une sous-ajustement du modèle aux données d'entraînement.

Les SVM possèdent enfin une complexité d'entraînement importante, entre $O(n^{2p})$ voir $O(n^{3p})$ pour un modèle avec n samples et p features. De plus, l'interprétation des résultats d'un SVM peut être complexe, car il est difficile de comprendre l'influence de chaque variable sur la décision finale.

En résumé, les SVM offrent un cadre puissant pour la classification et la régression en utilisant des hyperplans de décision optimaux et des kernels pour traiter des données non linéaires. Malgré certains inconvénients, ils restent très utilisés pour leur versatilité. [5]

3.1.2 AdaBoost

Le modèle d'IA Adaboost, également connu sous le nom de "AdaBoost" (Adaptive Boosting), est une méthode d'apprentissage supervisé utilisée pour la classification. L'idée principale derrière son fonctionnement est de combiner plusieurs classificateurs faibles pour former un classificateur fort. Les classificateurs faibles sont des modèles simples qui ont une performance légèrement meilleure que le hasard, comme des arbres de décision faibles.

Le processus d'Adaboost consiste à entraîner itérativement une séquence de classificateurs faibles sur des sous-ensembles d'entraînement pondérés. À chaque itération, les poids des échantillons d'entraînement sont ajustés pour donner plus d'importance aux exemples mal classés par les classificateurs précédents. Ainsi, les classificateurs faibles suivants se concentrent davantage sur les exemples difficiles à classer, améliorant progressivement les performances globales du modèle.

Les avantages d'Adaboost résident dans sa capacité à obtenir de très bons résultats de classification avec une combinaison de classificateurs faibles. Il peut être utilisé avec différents algorithmes de base et offre une grande flexibilité. De plus, Adaboost est résistant au sur-ajustement et peut gérer des ensembles de données bruités.

Cependant, Adaboost peut être sensible aux valeurs aberrantes dans les données, ce qui peut entraîner une mauvaise performance du modèle. De plus, le processus d'entraînement d'Adaboost peut être plus lent que celui d'autres algorithmes, car il nécessite plusieurs itérations pour construire un classificateur fort.

En résumé, Adaboost est un modèle d'IA puissant pour la classification, qui combine des classificateurs faibles pour former un classificateur fort. Il offre de bons résultats de classification, une flexibilité et une résistance au sur-ajustement. Cependant, il peut être sensible aux valeurs aberrantes et nécessiter un temps d'entraînement plus long.

Pour l'implémentation, nous avons utilisé la bibliothèque scikit-learn qui implémente l'algorithme AdaBoost-SAMME [6]

3.1.3 Extreme Gradient Boost

Le modèle XGBoost, abréviation de "Extreme Gradient Boosting", est une méthode d'apprentissage automatique qui appartient à la famille des modèles de gradient boosting. L'idée principale derrière son fonctionnement est de construire un modèle prédictif en combinant plusieurs modèles de prédiction faibles, tels que des arbres de décision, de manière itérative.

Le fonctionnement d'XGBoost repose sur l'optimisation d'une fonction de perte en utilisant le gradient boosting. À chaque itération, un nouvel arbre de décision est ajouté au modèle pour capturer les erreurs résiduelles du modèle précédent. L'algorithme recherche l'arbre optimal qui minimise la fonction de perte en utilisant une technique d'optimisation appelée "gradient boosting".

Les avantages d'XGBoost résident dans sa capacité à fournir des modèles de prédiction de haute qualité avec une grande précision. Il est capable de gérer efficacement des ensembles de données volumineux et complexes grâce à des techniques d'optimisation avancées et à un parallélisme efficace. De plus, XGBoost fournit des fonctionnalités telles que la gestion des valeurs manquantes et la régularisation pour prévenir le sur-ajustement.

Cependant, XGBoost peut nécessiter un réglage minutieux des hyperparamètres pour obtenir les meilleurs résultats, ce qui peut être un processus complexe et nécessiter une expertise supplémentaire. De plus, l'entraînement d'XGBoost peut être relativement lent en raison de la construction itérative de nombreux arbres de décision. [7] [8]

3.1.4 Multi-Layer Perceptron Classifier

Le modèle Multi-Layer Perceptron Classifier (MLPC), est une méthode d'apprentissage automatique basée sur les réseaux de neurones artificiels. L'idée principale derrière son fonctionnement est de construire un réseau de neurones à plusieurs couches pour effectuer la classification des données.

Le MLPC est composé de plusieurs couches de neurones, comprenant une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Chaque neurone dans les couches cachées est connecté à tous les neurones de la couche précédente et de la couche suivante, formant ainsi un réseau dense. Chaque neurone effectue une transformation non linéaire de l'entrée pondérée par des poids, généralement suivie d'une fonction d'activation.

L'entraînement du MLPC se fait par rétropropagation du gradient, où les poids des connexions entre les neurones sont ajustés pour minimiser la fonction de perte. Cela se fait en utilisant des techniques d'optimisation telles que la descente de gradient stochastique. Une fois le modèle entraîné, il peut être utilisé pour prédire la classe d'un nouvel échantillon en passant cet échantillon à travers le réseau de neurones.

Les avantages du MLPC résident dans sa capacité à modéliser des relations complexes et non linéaires entre les caractéristiques d'entrée et les classes cibles. Il peut apprendre à partir de grandes quantités de données et généraliser efficacement. De plus, le MLPC est capable de traiter des ensembles de données de grande dimension et de faire face à des problèmes de classification difficiles.

MLPC peut également nécessiter un réglage minutieux des hyperparamètres pour obtenir de bonnes performances et éviter le sur-ajustement. De plus, l'entraînement d'un MLPC peut être coûteux en termes de temps de calcul, en particulier avec de grandes architectures de réseau et des ensembles de données volumineux. Enfin, l'interprétation des résultats du MLPC peut être difficile en raison de la complexité du modèle. [9]

3.1.5 Random Forest

Le modèle Random Forest est une méthode d'apprentissage automatique basée sur l'ensemble d'arbres de décision. L'idée principale derrière son fonctionnement est de construire un grand nombre d'arbres de décision indépendants et de combiner leurs prédictions pour obtenir un résultat final.

La random forest fonctionne en créant un échantillon aléatoire avec remplacement à partir de l'ensemble d'entraînement pour chaque arbre de décision. Chaque arbre est construit en utilisant un sous-ensemble différent des caractéristiques d'entrée, ce qui favorise la diversité des arbres. Lors de la phase de prédiction, les prédictions de chaque arbre sont agrégées, par exemple en utilisant un vote majoritaire pour les problèmes de classification ou une moyenne pour les problèmes de régression.

Les avantages de la random forest résident dans sa capacité à fournir des modèles robustes et précis, avec une bonne généralisation. Elle est moins sujette au sur-ajustement que les arbres de décision individuels, car elle utilise des moyennes et des votes majoritaires pour prendre des décisions. De plus, la random forest est capable de gérer des ensembles de données de grande dimension et des caractéristiques non linéaires.

Cependant, la random forest peut être plus complexe à interpréter que les arbres de décision individuels en raison de la combinaison de plusieurs arbres. La complexité d'entraînement est de $O(t * u * n * \log(n))$ où t est le nombre d'arbres, u est le nombre de features et n le nombre de samples. Elle peut également être plus lente à entraîner et à prédire, en raison du grand nombre d'arbres et du calcul supplémentaire requis pour agréger les résultats. La prédiction se fait en $O(t * \log(n))$. [10]

3.2 Deep Learning

3.2.1 Sequential Model

Le modèle séquentiel de deep learning est une approche puissante utilisée dans le domaine de l'apprentissage automatique pour résoudre des tâches complexes telles que la reconnaissance d'images, la traduction automatique et la génération de texte. L'idée principale derrière son fonctionnement est de construire un réseau de neurones artificiels avec plusieurs couches (d'où le terme "deep"), permettant une représentation hiérarchique et progressive des données.

Dans un modèle séquentiel, les couches sont organisées en séquence, où chaque couche traite les informations reçues de la couche précédente. Les réseaux de neurones séquentiels les plus couramment utilisés sont les réseaux de neurones convolutifs (CNN) pour le traitement des images et les réseaux de neurones récurrents (RNN) pour les données séquentielles telles que les séquences de mots ou de phrases.

Les avantages du modèle séquentiel de deep learning résident dans sa capacité à apprendre des représentations de données hautement abstraites et dans sa capacité à capturer des motifs complexes et non linéaires. Les architectures de réseau profond permettent d'apprendre de manière automatique des caractéristiques et des structures pertinentes à partir des données, sans nécessiter une ingénierie manuelle des caractéristiques. De plus, ces modèles peuvent être entraînés sur de grandes quantités de données, ce qui améliore leurs performances.

Cependant, les modèles séquentiels de deep learning peuvent nécessiter une grande quantité de données d'entraînement pour atteindre de bonnes performances, et l'entraînement peut être coûteux en termes de temps et de ressources computationnelles. De plus, l'interprétation des résultats peut être complexe en raison de la nature non linéaire et des représentations abstraites apprises par le modèle. [11]

4 Evaluation

Dans cette partie, nous discuterons les différentes méthodes d'évaluations des modèles cité précédemment, et le choix de celles-ci dans notre problème.

4.1 Difference de l'évaluation entre classification multi-label et multi-class

La différence entre une classification multi-label et une classification multi-class est l'exclusivité. Une classification mutli-class n'autorise qu'une seule classification par sample, tandis qu'une classification multi-label permet plusieurs labels par sample. Dans le cas de notre application nous sommes dans du multi-class, où chaque sample correspond à un type de malware. [12]

Cette différence fondamentale influe donc forcément les méthodes d'évaluations engagées. Dans notre cas, nous allons regarder des méthodes qui se concentrent sur la détection d'une seule classe, sans prendre en compte les influences des autres sur notre choix. [13]

Nous prendrons donc principalement les valeurs suivantes pour créer nos méthodes d'évaluations :

- TP - True Positive : les prédictions positives justes.
- TN - True Negative : les prédictions négatives justes.
- FP - False Positive : les prédictions positives fausses.
- FN - False Negative : les prédictions négatives fausses.

Ces valeurs sont cependant justes seulement dans le cas où nous effectuons une classification sur 2 classes.

Dans un cas avec plus de deux classes, nous nous référons à la matrice de confusion de notre modèle. Celle ci décrit les prédictions attendues et obtenues par classes, et permet d'adapter les valeurs vues précédemment. Nous pouvons l'illustrer avec un exemple pour une classification à trois éléments.

Class / Prediction	Class1	Class2	Class3
Class1	T1	F1	F1
Class2	F2	T2	F2
Class3	F3	F3	T3

Table 1: Matrice de confusion pour une classification sur 3 labels distincts

Cette matrice met en évidence les prédictions réalisées par le modèle par rapport aux attentes. Dans ce modèle, il n'existe plus de FP et FN, mais seulement des valeurs justes ou fausses. Les valeurs diagonales a_{ij} avec $i = j$ représente les prédictions positives, tandis que les termes a_{ij} avec i différent de j représente les prédictions négatives. [14]

4.2 Accuracy

La première méthode pour réaliser une évaluation est l'accuracy de notre modèle. Celle ci représente le nombre de précisions correcte sur l'ensemble des prédictions réalisées.

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

Dans un modèle avec N classes, il est nécessaire d'adapter la formule précédente de la sorte en utilisant la matrice de confusion A :

$$Accuracy = \frac{\sum_{i=1}^N a_{ii}}{\sum_{i=1}^N \sum_{j=1}^N a_{ij}}$$

Elle présente un résultat simple à interpréter et qui permet d'avoir une idée de le fonctionnement général du modèle, mais qui ne permet pas de rentrer dans les détails du multi-class. En analysant l'ensemble des prédictions faites, il n'y a donc plus de distinctions, et il peut être compliqué de savoir comment améliorer le modèle.

4.3 Recall

Le recall est calculé en réalisant la fraction des TP par l'ensemble des prédictions positives du set de données initial.

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

Le but du recall est de mesurer la capacité du modèle à trouver tous les positifs dans le set originel. Il ne peut être adapté à du multi-class, dans quel cas on le fera alors pour chaque modèle individuellement, dans une classification en OneVsRest, pour se ramener à une problème de classification binaire. [14]

4.4 Precision

La précision est calculé en réalisant la fraction des TP par l'ensemble des prédictions positives du modèle.

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

Le but de la précision est de mesurer la proportion de prédiction que notre modèle a réaliser positives de manière adéquate, par rapport à l'ensemble de ses prédictions positives. Cela permet d'avoir une idée sur la confiance que l'on peut donner à notre modèle lorsque celui-ci réaliser une prédiction positive. [14]

De la même manière que pour le recall, cette valeur n'est pas adaptable directement pour du multi-class, et doit être calculé en OneVsRest, ramenant le problème à une classification binaire.

4.5 F1-Score

Le F1-Score est la moyenne harmonique du recall et de la précision, calculée de la manière suivante :

$$F1_{score} = \frac{2 * (Prec * Rec)}{Prec + Rec}$$

Cette valeur peut être vue comme une moyenne pondérée entre la précision et le recall, où les 2 ont le même poids. Le F1-score est égal à 1 si on a une précision et un recall parfait, et à 0 si soit le recall, soit la précision est nulle.

Cette valeur, utilisant la précision et le recall, doit ainsi être calculé en OneVsRest pour toutes les classes de notre problème.

4.6 Macro-average F1-Score

Le Macro-average F1-Score calcule la moyenne non pondérée des F1-Score pour chaque classe du modèle. Elle est calculée de la manière suivante :

$$MacroAverageF1Score = \frac{\sum_{i=1}^N F1Score_{class(i)}}{N}$$

Cette formule est adaptable au Recall et à la Précision également. Elle permet d'avoir une vue d'ensemble du modèle en utilisant les valeurs précédemment décrites.

4.7 Weighted-average F1-Score

Le Weighted-average F1-Score vise à donner une vue d'ensemble du modèle, de la même manière que le Macro-average F1-Score, mais en prenant en compte la taille des échantillons pour pondérer son calcul.

Chaque F1-Score est multiplié par sa proportion tel que :

$$WeightedF1Score = \frac{2 * (Prc(Weight) * Rec(Weight))}{(Prc(Weight) + Rec(Weight))}$$

avec :

$$Prc(Weight) = \frac{\sum_{i=1}^N Prc(class(i)) * class(i)}{\sum_{i=1}^N class(i)}$$

$$Rec(Weight) = \frac{\sum_{i=1}^N Rec(class(i)) * class(i)}{\sum_{i=1}^N class(i)}$$

Cette mesure permet ainsi de pondérer l'évaluation de notre modèle en fonction des données d'entrées, et ainsi tenter d'avoir une mesure relativement fiable pour son évaluation.

4.8 Courbe ROC et score AUC

Le ROC (Receiver Operating Characteristic) est une courbe graphique utilisée pour évaluer et visualiser les performances d'un modèle de classification binaire. L'idée principale derrière le ROC est de représenter le taux de vrais positifs (sensibilité) par rapport au taux de faux positifs (1 - spécificité) à différents seuils de classification.

Le ROC est construit en calculant le taux de vrais positifs (VPR) et le taux de faux positifs (FPR) à différents seuils de classification. Un seuil est utilisé pour décider quelle classe prédite par le modèle sera considérée comme positive et quelle classe sera considérée comme négative. En faisant varier le seuil, on peut observer comment les performances du modèle évoluent.

La courbe ROC est créée en traçant le FPR (False Positive Rate) en fonction du TPR ou VPR (True/Vrai positif rate) pour chaque seuil. Un modèle idéal qui prédit parfaitement les classes aurait un VPR de 1 et un FPR de 0, ce qui correspondrait à un point situé dans le coin supérieur gauche du graphique. La ligne diagonale du graphique représente les performances aléatoires d'un modèle.

Les avantages du ROC résident dans sa capacité à fournir une évaluation globale des performances d'un modèle de classification binaire, indépendamment du seuil de classification choisi. La courbe ROC permet de visualiser les compromis entre le taux de vrais positifs et le taux de faux positifs, ce qui est particulièrement utile lorsque les classes sont déséquilibrées. [15]

De plus, les courbes ROC peuvent être mises en relation avec les histogrammes de séparation de classes. On observe que plus ceux-ci sont distincts, plus les courbes ROC seront bonnes. [16]

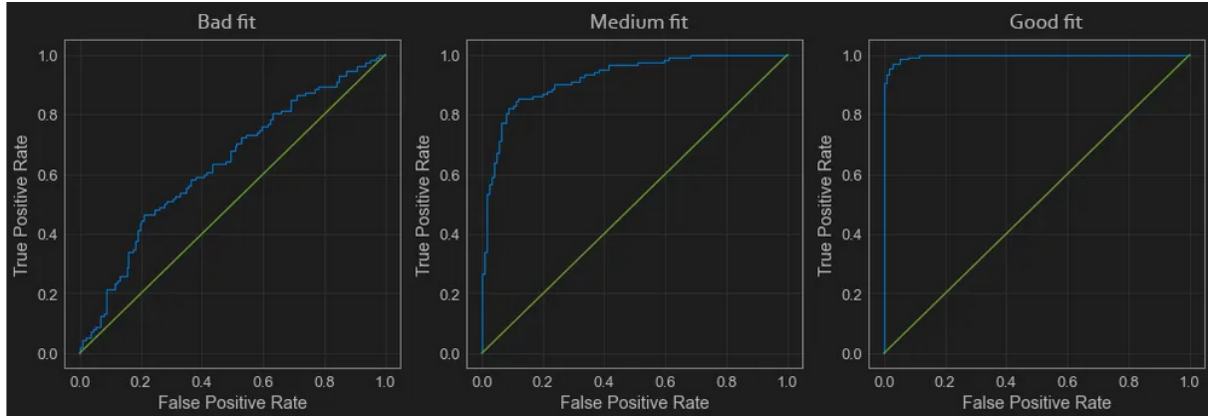


Figure 6: Exemples de courbes ROC pour différents fit

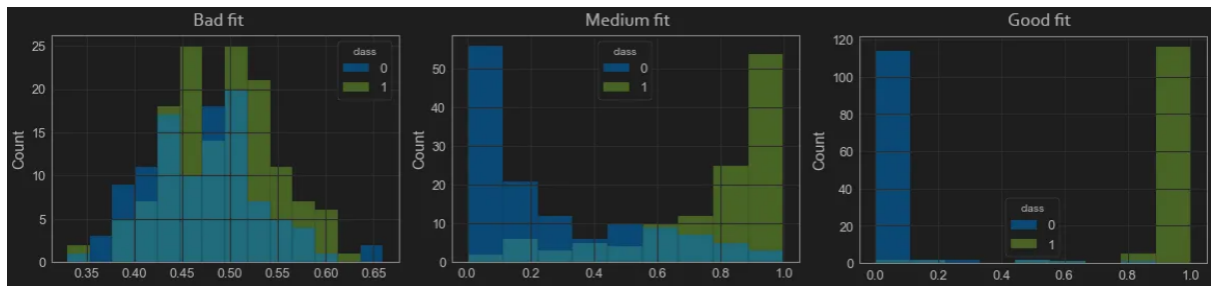


Figure 7: Exemples d'histogrammes pour différents fit

Le score AUC-ROC (Area Under the ROC Curve) ou AUC peut être calculé pour quantifier les performances du modèle de manière agrégée. Celui ci sera de 0.5 pour un modèle aléatoire, représentant l'air sous la droite verte des courbes ROC, et de 1 si le modèle est parfait. [17]

Ces valeurs ne sont cependant calculable que pour de la classification binaire. Pour la classification multiclass, il est alors nécessaire de calculer les courbes ROC et les scores AUC soit en OneVersusRest, soit en OneVersusOne. [18]

5 Resultats

Dans cette section nous présenterons rapidement les résultats obtenus par les différents modèles en traitant la classification binaire et multiclass séparément. Nous analyserons et discuterons de ces résultats dans la section suivante.

5.1 Détection

5.1.1 Classification Reports

Le premier tableau offre une vue générale de la performance de chacun des algorithmes à des fins de comparaisons. Les tableaux suivants détaillent les résultats pour chaque modèle, permettant une analyse approfondie.

	Accuracy	Macro F1-Score	Weighted F1-Score
Random Forest	0.98872	0.86297	0.98748
SVM	0.98621	0.81371	0.98374
XGBoost	0.99009	0.88147	0.98909
MLPC	0.98678	0.83640	0.98517
Adaboost	0.98325	0.78751	0.98095
DL	0.98564	0.81105	0.98332

Table 2: Résultats généraux des algorithmes mis en place pour la détection

	Precision	Recall	F1-Score
No Malware	0.94406	0.59735	0.73171
Malware	0.98946	0.99906	0.99424
Accuracy	0.98872		
Macro-average	0.96676	0.79820	0.86297
Weighted-average	0.98829	0.98872	0.98748

Table 3: Détail des résultats de Random Forest pour la détection

	Precision	Recall	F1-Score
No Malware	1.00000	0.46460	0.63444
Malware	0.98605	1.00000	0.99297
Accuracy	0.98621		
Macro-average	0.99302	0.73230	0.81371
Weighted-average	0.98640	0.98621	0.98374

Table 4: Détail des résultats de SVM pour la détection

	Precision	Recall	F1-Score
No Malware	0.81102	0.45575	0.58357
Malware	0.98578	0.99719	0.99145
Accuracy	0.98325		
Macro-average	0.89840	0.72647	0.78751
Weighted-average	0.98128	0.98325	0.98095

Table 5: Détail des résultats de Adaboost pour la détection

	Precision	Recall	F1-Score
No Malware	0.96644	0.63717	0.76800
Malware	0.99049	0.99942	0.99494
Accuracy	0.99009		
Macro-average	0.97847	0.81829	0.88147
Weighted-average	0.98988	0.99009	0.98909

Table 6: Détail des résultats de XGBoost pour la détection

	Precision	Recall	F1-Score
No Malware	0.90441	0.54425	0.67956
Malware	0.98808	0.99848	0.99325
Accuracy	0.98678		
Macro-average	0.94625	0.77136	0.83640
Weighted-average	0.98592	0.98678	0.98517

Table 7: Détail des résultats de MLPC pour la détection

	Precision	Recall	F1-Score
No Malware	0.93860	0.47345	0.62941
Malware	0.98626	0.99918	0.99268
Accuracy	0.98564		
Macro-average	0.96243	0.73632	0.81105
Weighted-average	0.98503	0.98564	0.98332

Table 8: Détail des résultats de DL pour la détection

5.1.2 Courbes ROC et scores AUC

Les graphiques suivant présentent les histogrammes, courbes ROC et scores AUC obtenus pour chaque modèle.

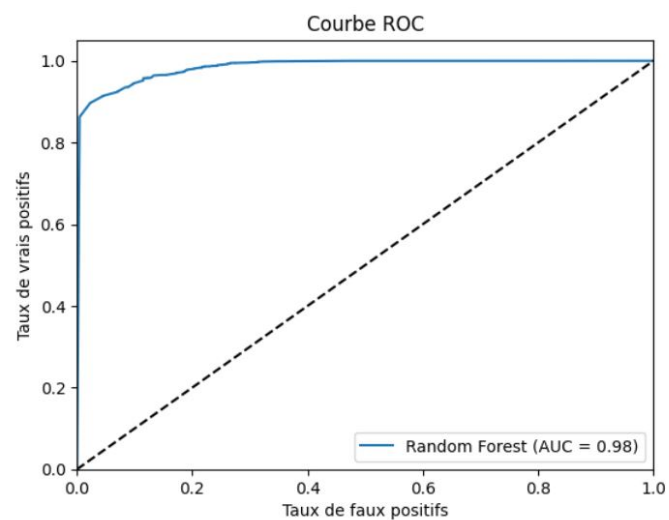


Figure 8: Courbe ROC et score AUC pour Random Forest

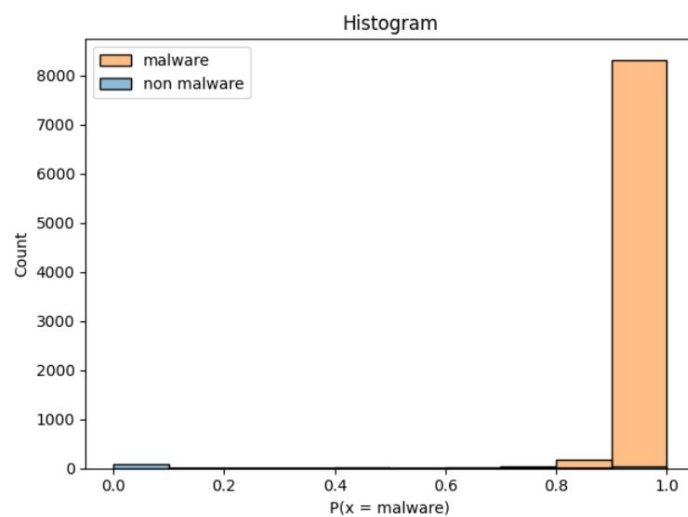


Figure 9: Histogramme pour Random Forest

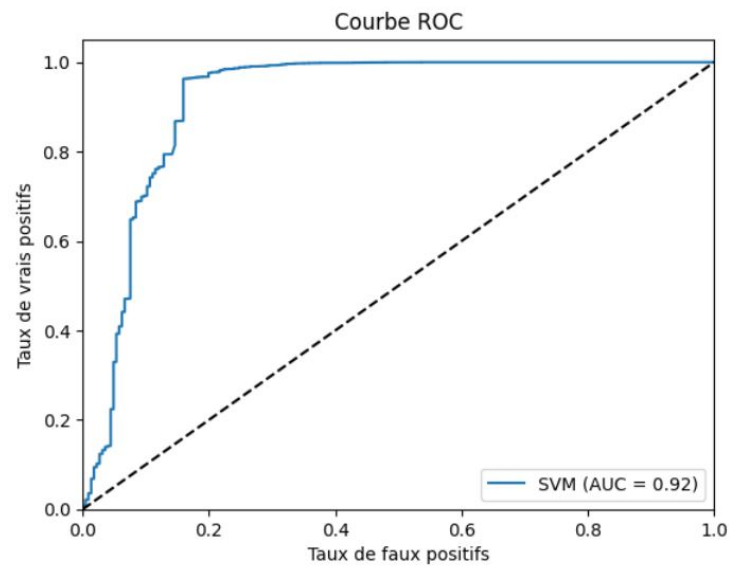


Figure 10: Courbe ROC et score AUC pour SVM

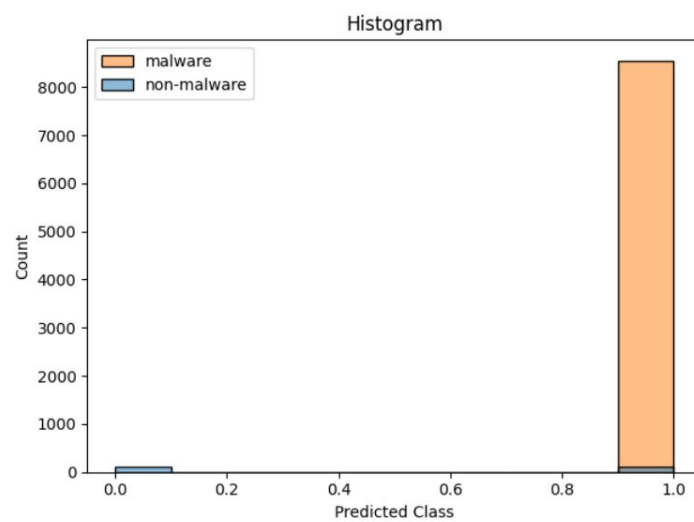


Figure 11: histogramme pour SVM

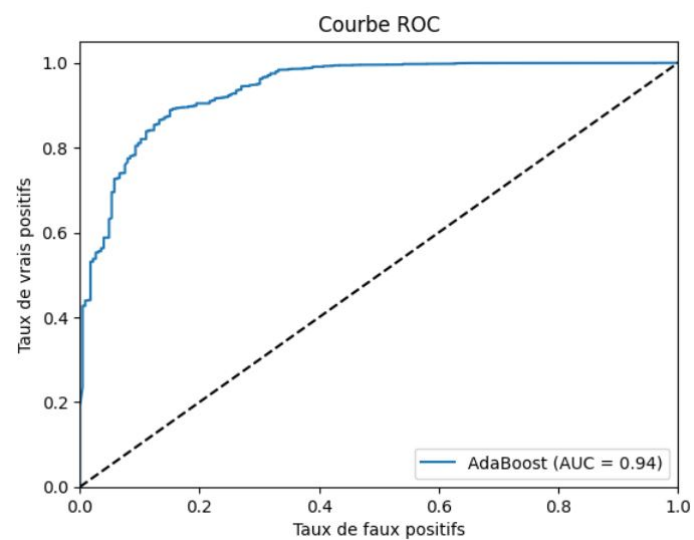


Figure 12: Courbe ROC et score AUC pour AdaBoost

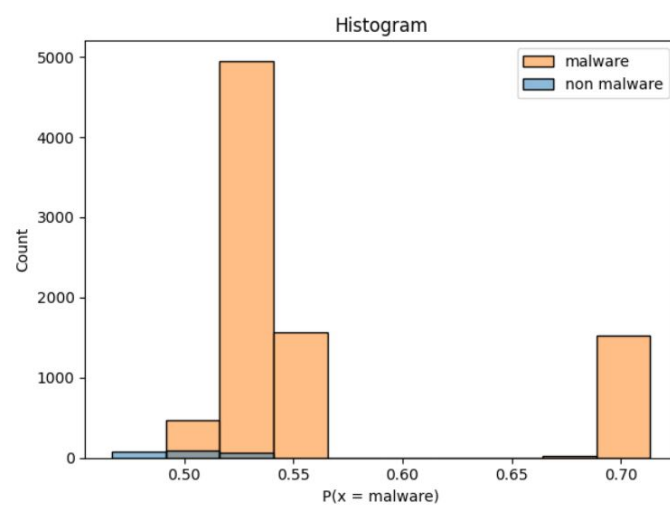


Figure 13: histogramme pour AdaBoost

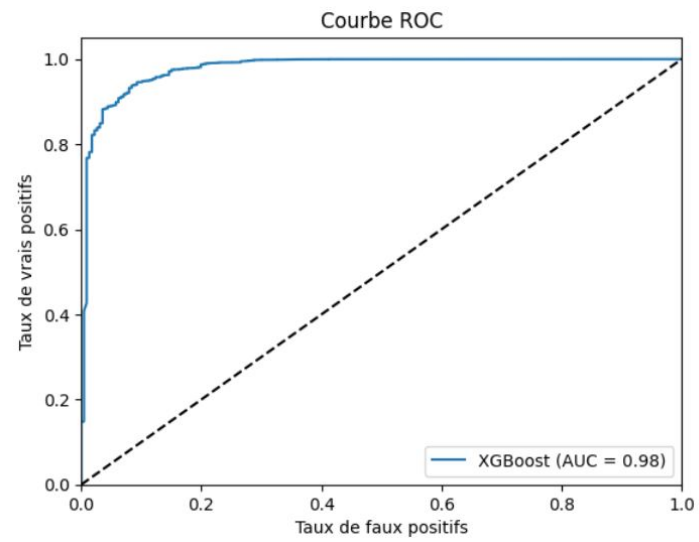


Figure 14: Courbe ROC et score AUC pour XGBoost

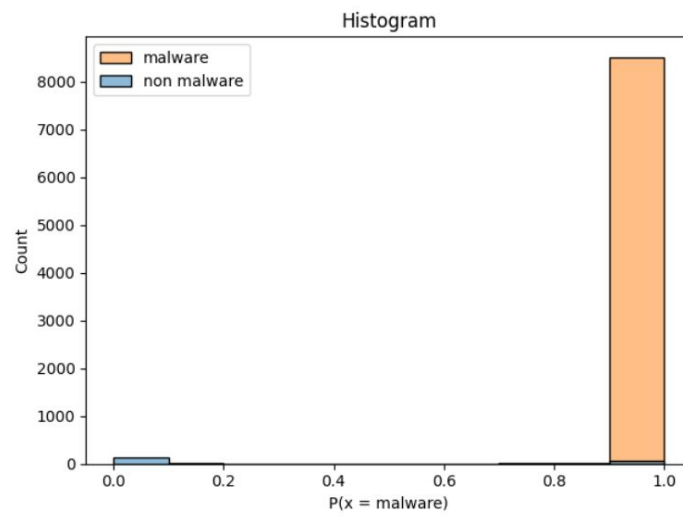


Figure 15: histogramme pour XGBoost

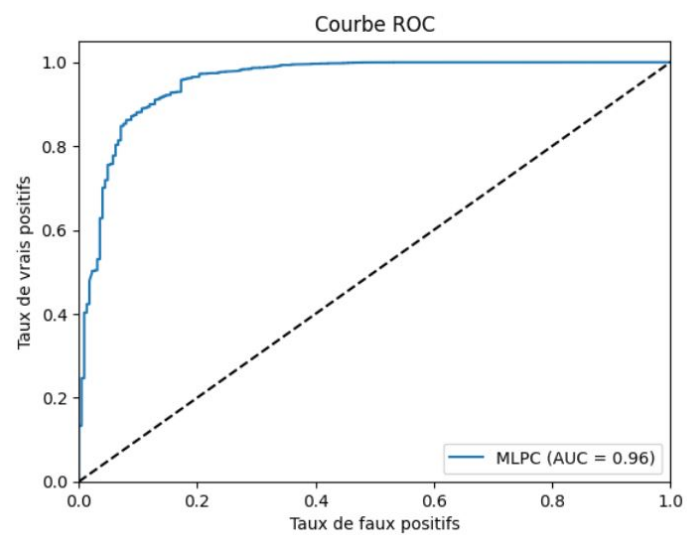


Figure 16: Courbe ROC et score AUC pour MLPC

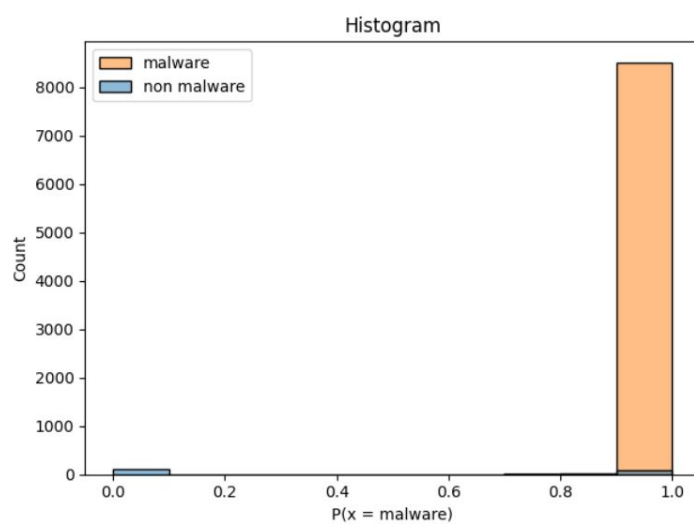


Figure 17: histogramme pour MLPC

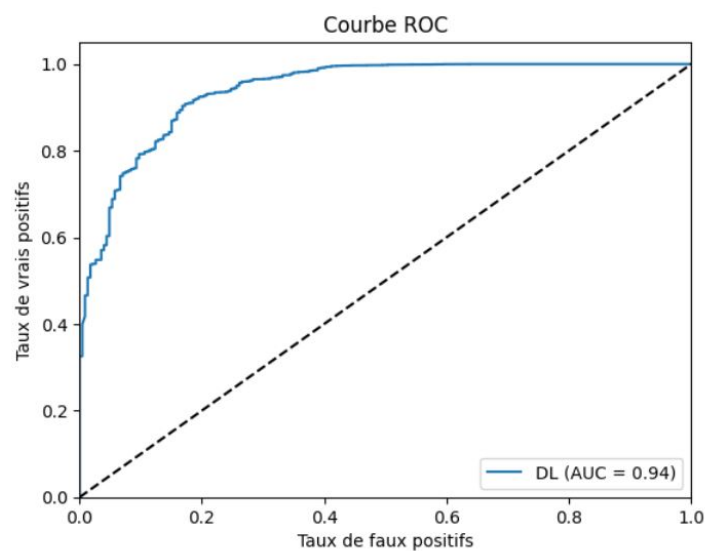


Figure 18: Courbe ROC et score AUC pour le modèle de Deep Learning

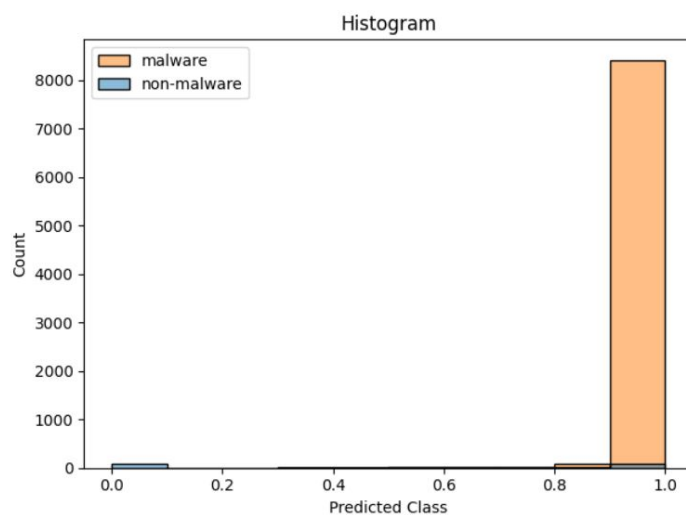


Figure 19: histogramme pour le modèle de Deep Learning

Method	AUC Score
Random Forest	0.98
AdaBoost	0.94
XGBoost	0.98
SVM	0.92
MLPC	0.96
DL	0.94

Table 9: Résumé des scores AUC

5.2 Classification

5.2.1 Classification Reports

Le premier tableau offre une vue générale de la performance de chacun des algorithmes à des fins de comparaisons. Les tableaux suivants détaillent les résultats pour chaque modèle, permettant une analyse approfondie.

	Accuraccy	Macro F1-Score	Weighted F1-Score
Random Forest	0.69691	0.70931	0.69644
Adaboost	0.44655	0.43540	0.43306
XGBoost	0.69691	0.71109	0.69970
MLPC	0.48594	0.50256	0.48286
SVM	0.17581	0.09710	0.10536
DL	0.09001	0.12248	0.13382

Table 10: Résultats généraux des algorithmes mis en place pour la classification

	Precision	Recall	F1-Score
Adware	0.00000	0.00000	0.00000
Backdoor	0.00000	0.00000	0.00000
Downloader	0.14565	0.97487	0.25343
Dropper	0.83333	0.02762	0.05348
Spyware	0.80000	0.07407	0.13559
Trojan	0.00000	0.00000	0.00000
Virus	0.82609	0.09744	0.17431
Worms	0.51282	0.09479	0.16000
Accuracy	0.17581		
Macro-average	0.38974	0.15860	0.09710
Weighted-average	0.40697	0.17581	0.10536

Table 11: Détail des résultats de SVM pour la classification

	Precision	Recall	F1-Score
Adware	0.54412	0.52113	0.53237
Backdoor	0.46207	0.33005	0.38506
Downloader	0.50198	0.63819	0.56195
Dropper	0.42784	0.45856	0.44267
Spyware	0.20455	0.11111	0.14400
Trojan	0.28111	0.30500	0.29257
Virus	0.64655	0.76923	0.70258
Worms	0.40889	0.43602	0.42202
Accuracy	0.44655		
Macro-average	0.43464	0.44616	0.43540
Weighted-average	0.43001	0.44655	0.43306

Table 12: Détail des résultats de Adaboost pour la classification

	Precision	Recall	F1-Score
Adware	0.93651	0.83099	0.88060
Backdoor	0.74737	0.69951	0.72265
Downloader	0.84066	0.76884	0.80315
Dropper	0.61929	0.67403	0.64550
Spyware	0.51741	0.64198	0.57300
Trojan	0.52284	0.51500	0.51889
Virus	0.83085	0.85641	0.84343
Worms	0.73822	0.66825	0.70149
Accuracy	0.69691		
Macro-average	0.71914	0.70688	0.71109
Weighted-average	0.70588	0.69691	0.69970

Table 13: Détail des résultats de XGBoost pour la classification

	Precision	Recall	F1-Score
Adware	0.87719	0.70423	0.78125
Backdoor	0.46222	0.51232	0.48598
Downloader	0.66667	0.62312	0.64416
Dropper	0.40000	0.47514	0.43434
Spyware	0.34426	0.25926	0.29577
Trojan	0.33880	0.31000	0.32376
Virus	0.54098	0.67692	0.60137
Worms	0.47895	0.43128	0.45387
Accuracy	0.48594		
Macro-average	0.51363	0.49903	0.50256
Weighted-average	0.48612	0.48594	0.48286

Table 14: Détail des résultats de MLPC pour la classification

	Precision	Recall	F1-Score
Adware	0.91176	0.87324	0.89209
Backdoor	0.75521	0.71429	0.73418
Downloader	0.79459	0.73869	0.76562
Dropper	0.60550	0.72928	0.66165
Spyware	0.55367	0.60494	0.57817
Trojan	0.56322	0.49000	0.52406
Virus	0.78673	0.85128	0.81773
Worms	0.72589	0.67773	0.70098
Accuracy	0.69691		
Macro-average	0.71207	0.70993	0.70931
Weighted-average	0.69949	0.69691	0.69644

Table 15: Détail des résultats de Random Forest pour la classification

	Precision	Recall	F1-Score
Adware	0.00000	0.00000	0.00000
Backdoor	0.00000	0.00000	0.00000
Downloader	0.77451	0.39698	0.52492
Dropper	0.00000	0.00000	0.00000
Spyware	0.50000	0.06173	0.10989
Trojan	0.00000	0.00000	0.00000
Virus	0.78261	0.18462	0.29876
Worms	1.00000	0.02370	0.04630
Accuracy	0.09001		
Macro-average	0.38214	0.08338	0.12248
Weighted-average	0.42105	0.09142	0.13382

Table 16: Détail des résultats de DL pour la classification

5.2.2 Scores AUC et courbes ROC

Les graphiques suivants présentent les histogrammes, courbes ROC et scores AUC obtenus pour chaque modèle. Ils ont été calculé en OneVersusRest pour permettre une vision relativement synthétique des résultats.

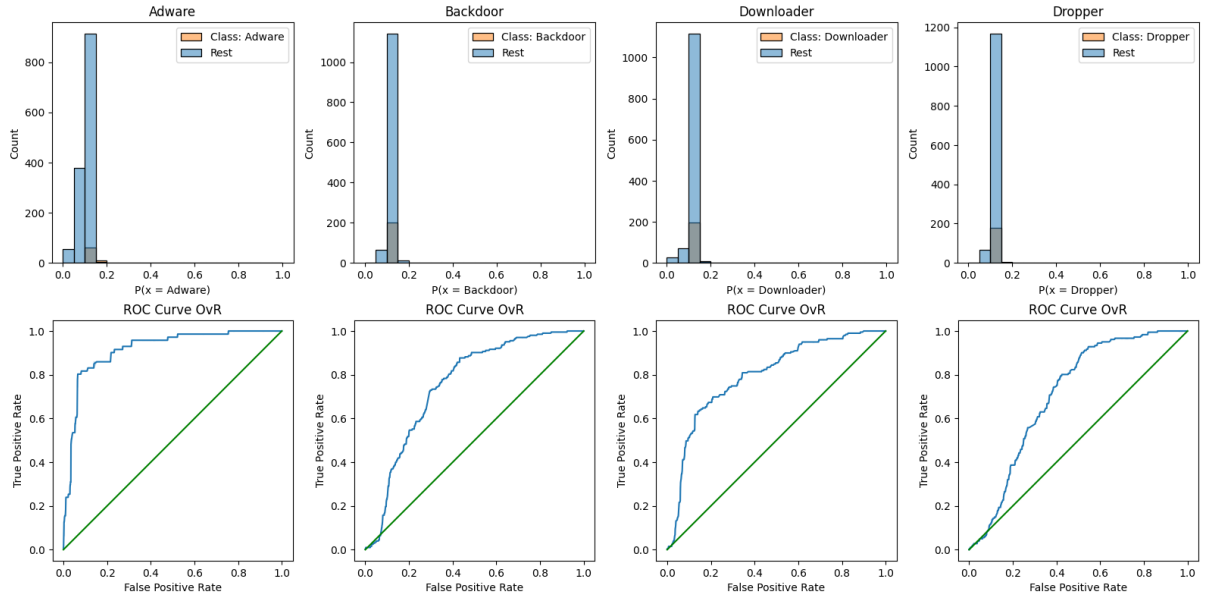


Figure 20: Courbes ROC et histogrammes pour Adaboost - partie 1

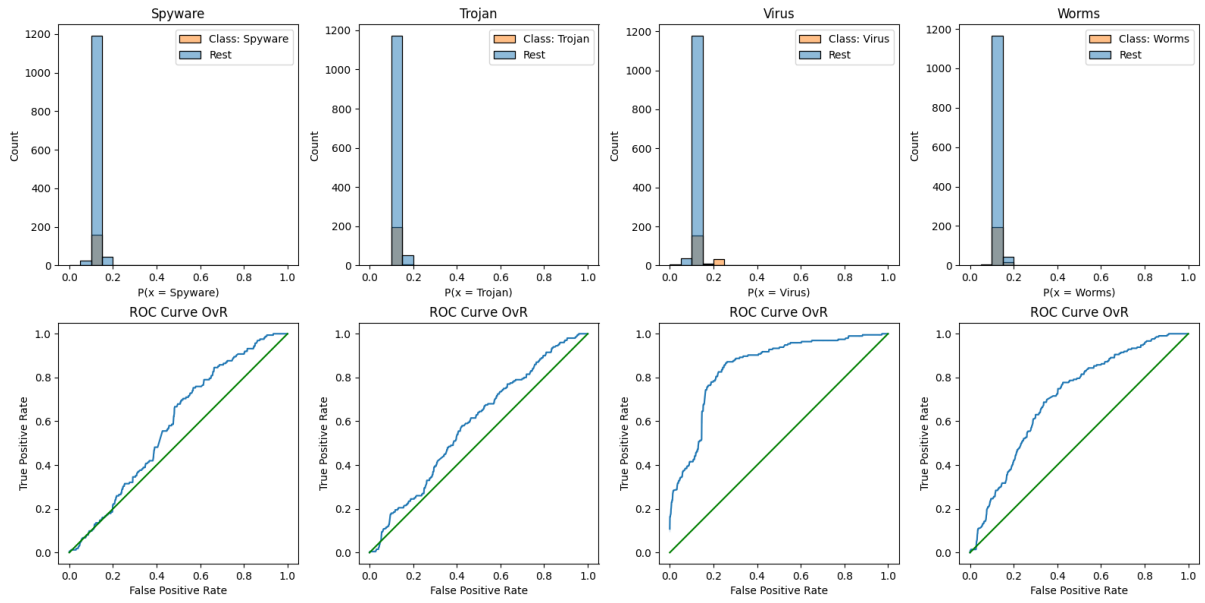


Figure 21: Courbes ROC et histogrammes pour Adaboost - partie 2

Class	AUC
Adware	0.9158
Backdoor	0.7549
Downloader	0.7961
Dropper	0.7085
Spyware	0.5836
Trojan	0.5852
Virus	0.8477
Worms	0.7056
Average	0.7372

Table 17: Scores AUC en OvR pour AdaBoost

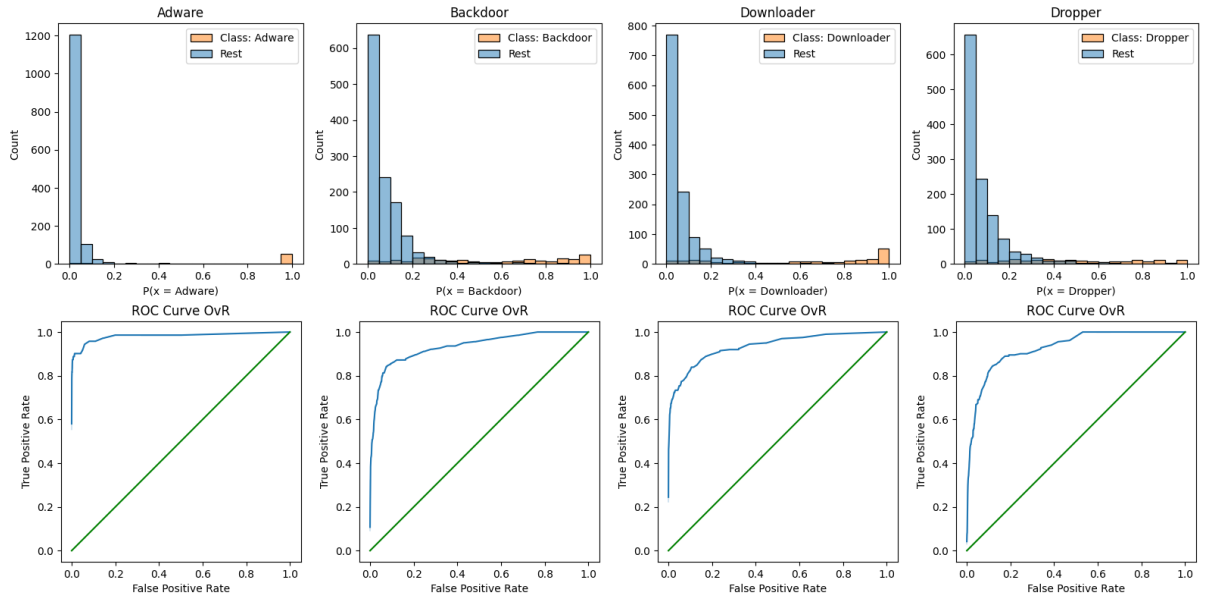


Figure 22: Courbes ROC et histogrammes pour Random Forest - partie 1

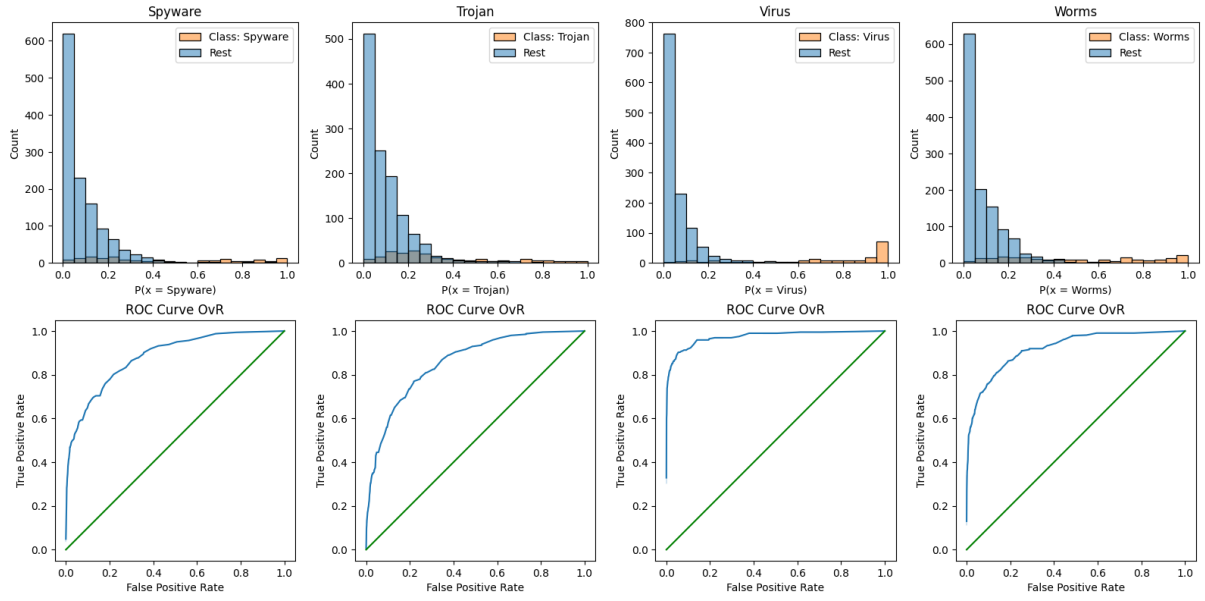


Figure 23: Courbes ROC et histogrammes pour Random Forest - partie 2

Class	AUC
Adware	0.9815
Backdoor	0.9336
Downloader	0.9342
Dropper	0.9273
Spyware	0.8814
Trojan	0.8543
Virus	0.9735
Worms	0.9207
Average	0.9258

Table 18: Scores AUC en OvR pour Random Forest

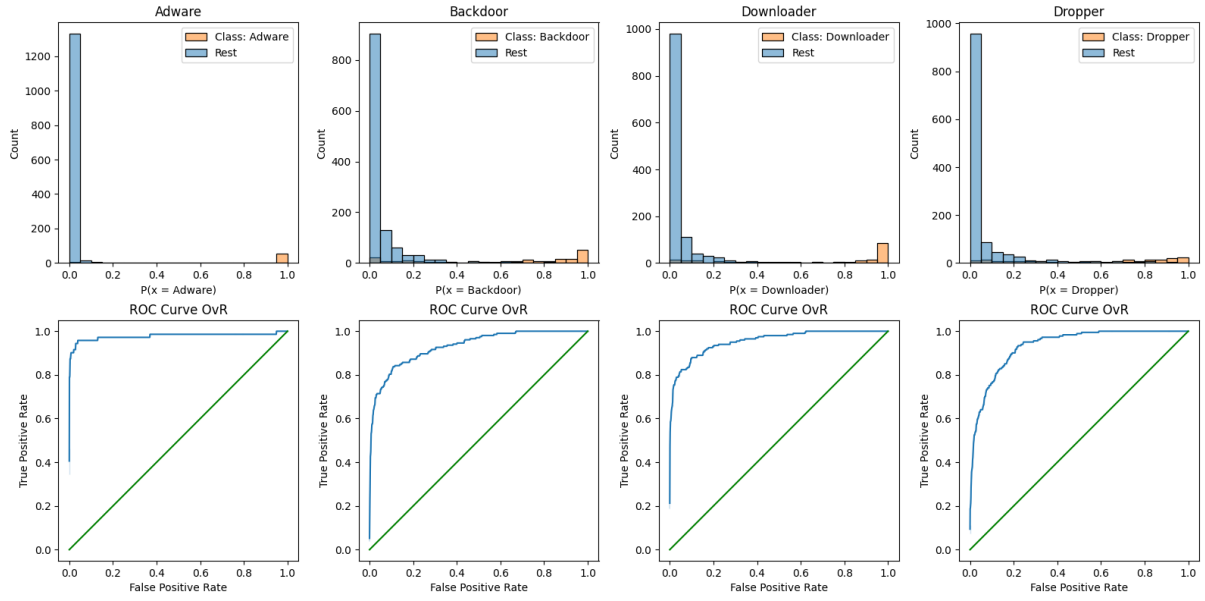


Figure 24: Courbes ROC et histogrammes pour XGBoost - partie 1

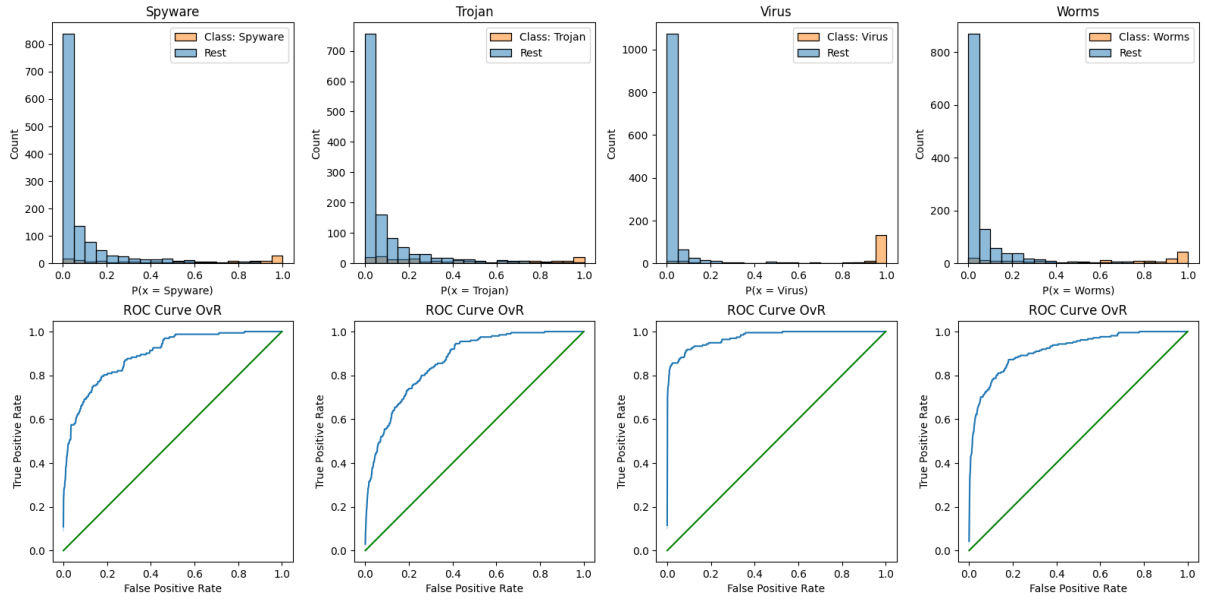


Figure 25: Courbes ROC et histogrammes pour XGBoost - partie 2

Class	AUC
Adware	0.9775
Backdoor	0.9308
Downloader	0.9547
Dropper	0.9329
Spyware	0.8939
Trojan	0.8633
Virus	0.9728
Worms	0.9156
Average	0.9302

Table 19: Scores AUC en OvR pour XGBoost

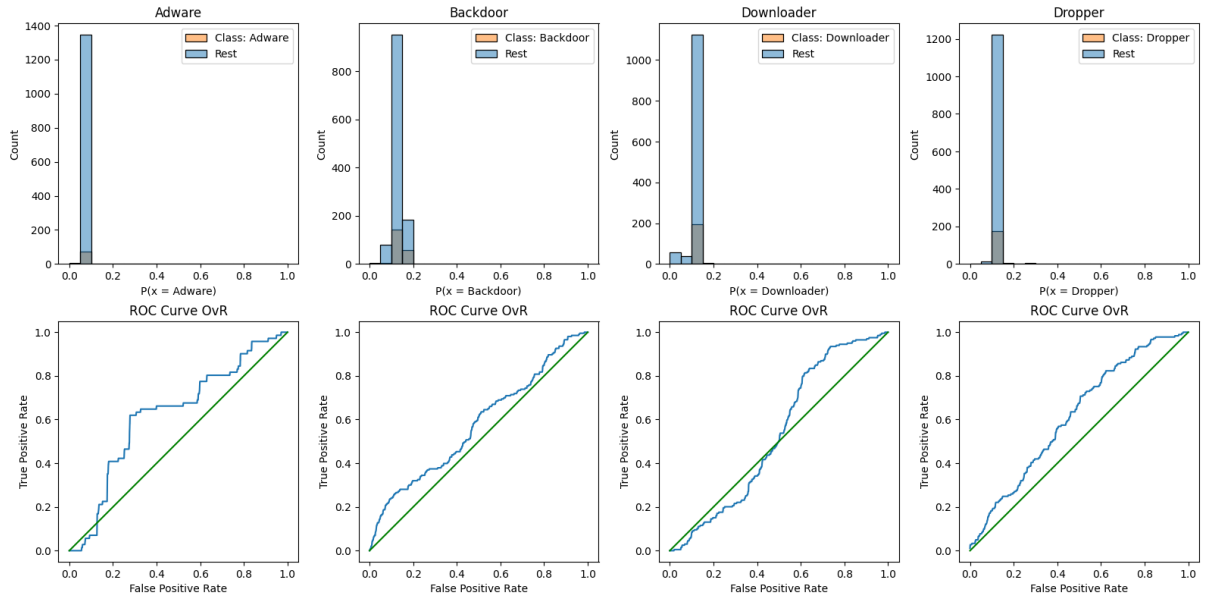


Figure 26: Courbes ROC et histogrammes pour SVM - partie 1

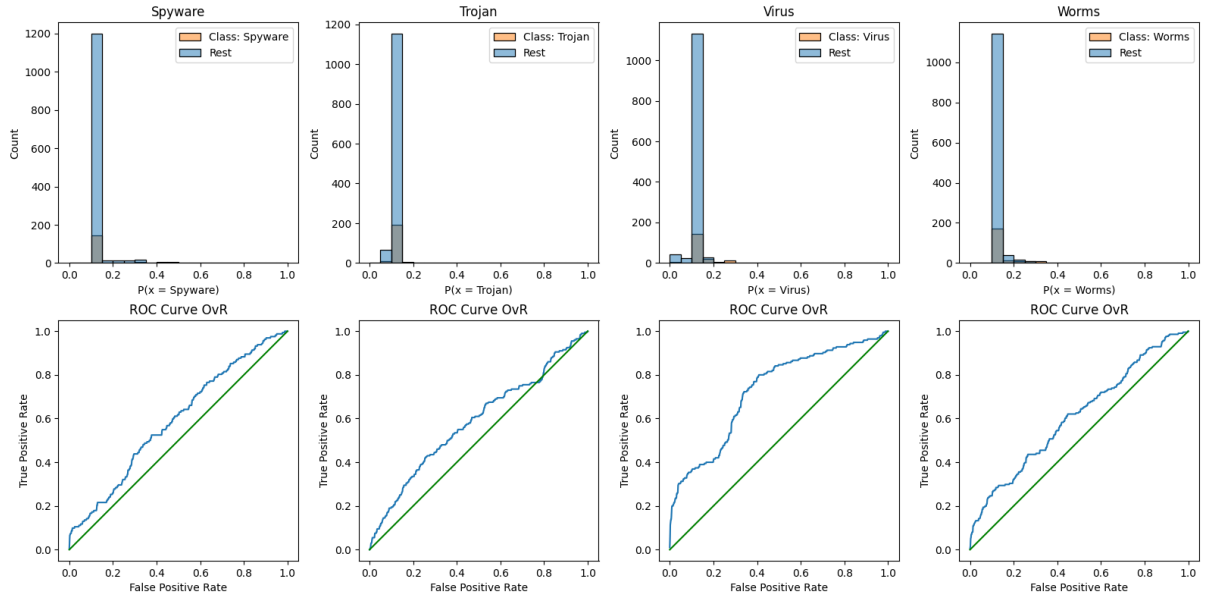


Figure 27: Courbes ROC et histogrammes pour SVM - partie 2

Class	AUC
Adware	0.6248
Backdoor	0.5752
Downloader	0.5345
Dropper	0.6159
Spyware	0.5904
Trojan	0.5856
Virus	0.7266
Worms	0.6110
Average	0.6080

Table 20: Scores AUC en OvR pour SVM

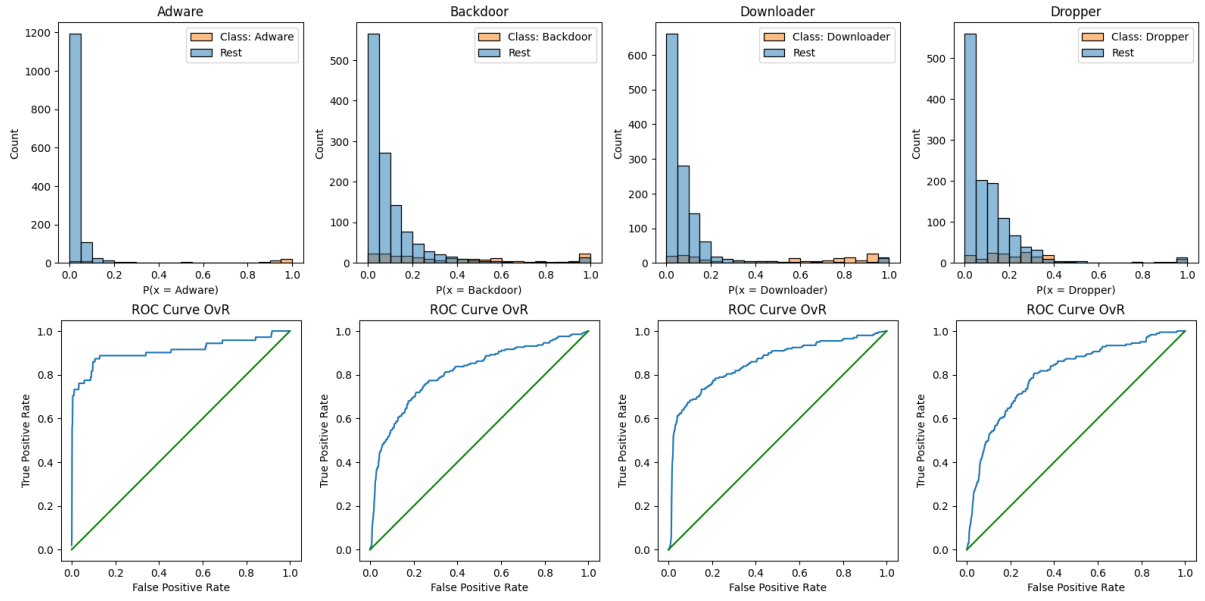


Figure 28: Courbes ROC et histogrammes pour MLPC - partie 1

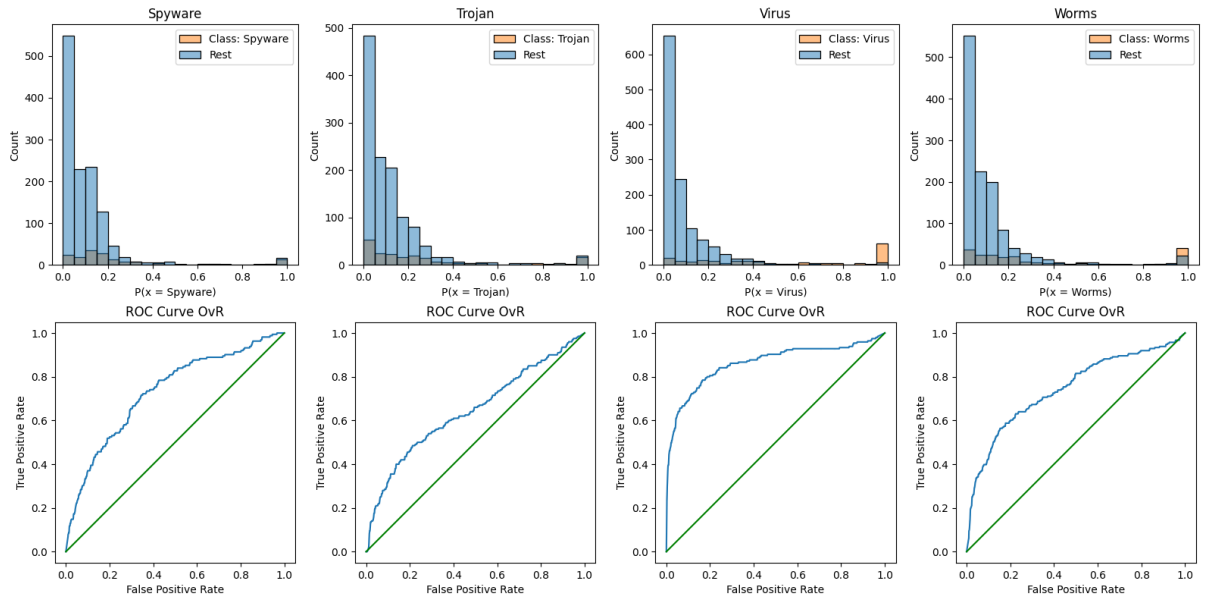


Figure 29: Courbes ROC et histogrammes pour MLPC - partie 2

Class	AUC
Adware	0.9095
Backdoor	0.8096
Downloader	0.8529
Dropper	0.8032
Spyware	0.7297
Trojan	0.6454
Virus	0.8618
Worms	0.7436
Average	0.7945

Table 21: Scores AUC en OvR pour MLPC

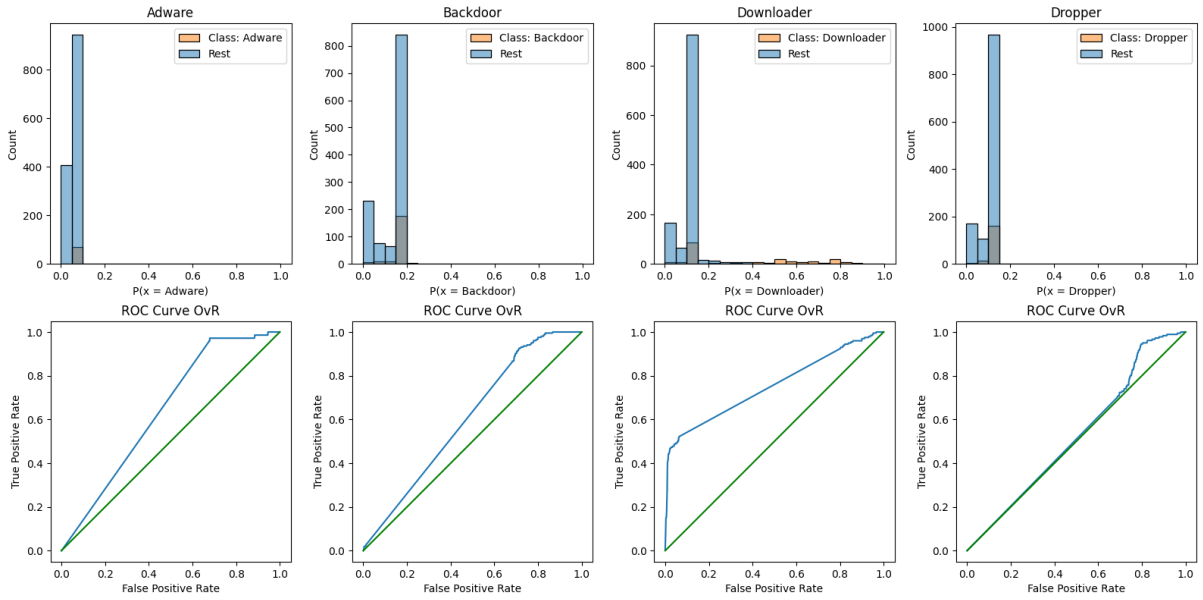


Figure 30: Courbes ROC et histogrammes pour DL - partie 1

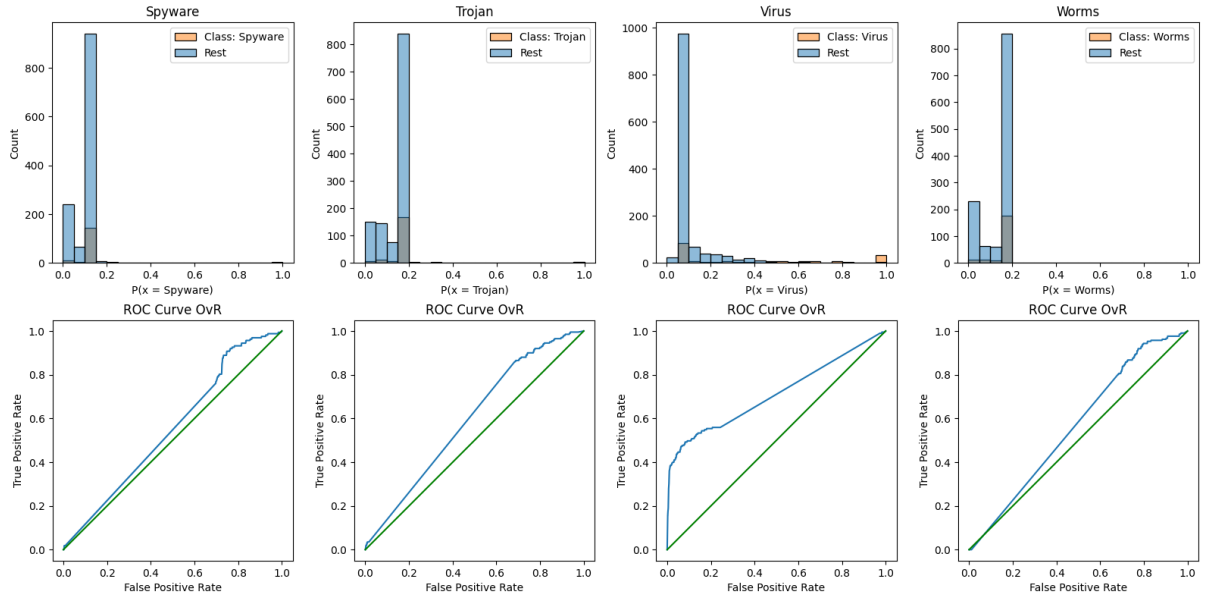


Figure 31: Courbes ROC et histogrammes pour DL - partie 2

Class	AUC
Adware	0.6403
Backdoor	0.6073
Downloader	0.7528
Dropper	0.5269
Spyware	0.5535
Trojan	0.5968
Virus	0.7113
Worms	0.5657
Average	0.6193

Table 22: Scores AUC en OvR pour DL

6 Discussion

Cette section vise à analyser les résultats obtenus pendant ce projet, de tenter de comprendre certaines particularités observées, et chercher des pistes d'améliorations pour poursuivre ce projet.

6.1 Détection

Le dataset sur lequel nous avons travaillé présente un déséquilibre important. Les valeurs de malwares sont beaucoup plus présentes que les valeurs saines. Nous analyserons donc en particulier l'accuracy du modèle et son weighted-F1-Score. Ceci est du au fait que

nous cherchons à détecter les malwares, et donc souhaitons apporter une importance proportionnée aux deux classes.

Lors de l'analyse des résultats, on remarque que les modèles présente tous des valeurs assez proches dans ces deux catégories. On peut observer des performances légèrement meilleurs pour XGBoost, et légèrement inférieur pour Adaboost.

De plus, le modèle utilisant du DeepLearning ne présente pas des performances différentes du machine learning. Il est dans la moyenne des résultats analysés.

Si on pousse désormais l'analyse sur le Macro-F1-Score, on s'aperçoit de différences assez nettes entre les algorithmes, mais qui suivent le rang des valeurs explicitées précédemment. Sous ce prisme on peut alors dire que XGBoost présente des résultats supérieurs au reste des algorithmes, avec près de 0.02 points d'avance sur Random Forest, second meilleur, et 0.1 point d'avance sur Adaboost, le moins performant.

Si enfin on regarde en détail les résultats obtenus pour les algorithmes, on s'aperçoit qu'ils ont tous un score assez faible pour le recall des samples sains. Ce problème étant partagé à tous les modèles, cela vient probablement d'un déséquilibre trop important dans le dataset d'entraînement. Il pourrait alors être possibles de mettre en place des techniques de rééquilibrage des données, tel aue la fusion de plusieurs datasets différents, pour observer les résultats et valider ou non cette hypothèse.

L'analyse des histogrammes confirme les observations précédentes. Pour certains modèles, comme Adaboost, on peut voir que la séparation des deux classes n'est pas effectué. Les bons scores AUC en apparence sont alors due au déséquilibre du dataset, ce qui est clairement visible dans les histogrammes.

6.2 Classification

Pour réaliser l'évaluation de cette problématique, nous nous reposerons principalement sur l'accuracy et le Macro-F1-Score.

Dans le cadre de la classification des données, on peut observer de grandes disparités entre les différents modèles. On peut les séparer en trois catégories distinctes.

- Random Forest et XGBoost.

Les deux présentent des résultats similaires avec 0,69691 d'accuracy et près de 0,71 de Macro F1-Score. Il est intéressant de constater que ces deux modèles ont également eu du mal à différencier les mêmes catégories, avec des scores de recall et de précision faibles pour Trojan, Spyware et Dropper en particulier.

- AdaBoost et MLPC.

De la même manière, ces deux modèles présentent des résultats proches d'accuracy et de Macro F1-Score, chacun autour de 0,45. De la même manière que pour les précédents modèles, les catégories Trojan, Spyware et Dropper ont particulièrement posés problèmes, avec des F1-scores faibles dans chacune des catégories.

- SVM et DL

SVM et DL présentent des résultats complètement en décalage avec les autres modèles, présentant des valeurs abérantes dans plusieurs catégories. Ce dernier semble avoir attribué la prédiction Downloader à la plupart de ces prédictions, sans que nous puissions en identifier la raison. Ceci est identifiable au recall très haut et à la précision très faible de la catégorie. Il est probable que la paramétrisation des modèles soit en défaut ici. Aussi bien le choix du kernels que les hyperparamètres pourrait êtres remis en question, et une analyse approfondie des données pourrait permettre d'identifier les valeurs adéquates.

En observant en détail les courbes ROC obtenus, on complète les observations précédentes. XGBoost et Random Forest retrouvent des Average AUC élevés.

Une hiérarchie se dessine plus clairement entre MLPC, Adaboost. SVM et DL retrouvent des courbes et des scores proches de l'aléatoire.

Enfin il est intéressant de regarder en détail les résultats du DL. Celui-ci ne possède pas, contrairement aux autres modèles, des scores AUC élevé et faibles pour certaines classes. Ils sont relativement homogènes, et les courbes ROC sont particulièrement différentes. Les histogrammes de répartitions sont similaires entre toutes les classes, et on observe donc que le modèle n'a pas réussi à réaliser une distinction importante entre elles.

Enfin, une analyse plus approfondie des histogrammes des classes Trojan, Spyware et Dropper ne possèdent pas de distinctions importantes par rapport au reste, et on retrouve ainsi les difficultés de classification observées précédement.

En résumé, nous pouvons observer que les algorithmes semblent se comporter de deux manières distinctes, en mettant à part SVM. On observe pour Random Forest et XGBoost des valeurs satisfaisantes pour une application de classification sur 8 classes distinctes. Le dataset d'origine possédant un nombre d'entrées assez faibles, environs 1000 par catégories, on peut facilement imaginer une amélioration possible à ce niveau.

Cependant, il est surprenant de voir que tous les algorithmes ont eu des scores faibles pour les catégories Trojan, Spyware et Dropper, et nous allons tenter, grâce à une analyse des données d'entrée, d'expliquer ces résultats.

6.2.1 Analyse des données

Lors de cette analyse de données, nous allons tenter de mettre en avant les différences principales entre les différentes catégories, en analysants plusieurs de leurs propriétés.

La premiere que nous allons regarder, est le nombre de samples par classes.

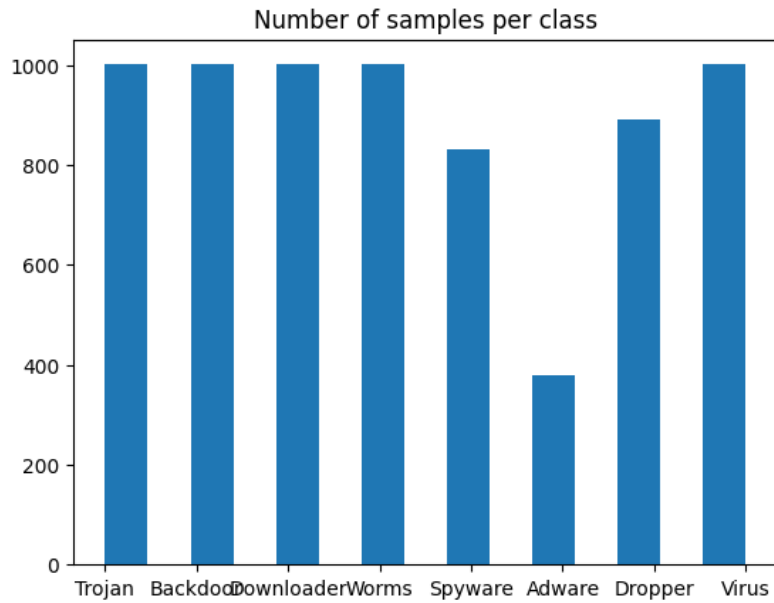


Figure 32: Nombre de samples par classe

On peut voir que 5 classes sont équitablement représentées, avec de plus Spyware et Dropper légèrement en retrait, et Adware sous représentées. Cela nous permet de voir que le dataset est assez petit en taille, et ne permettra pas un entraînement parfait.

Ensuite nous allons analyser la longueur moyenne d'un sample par classe.

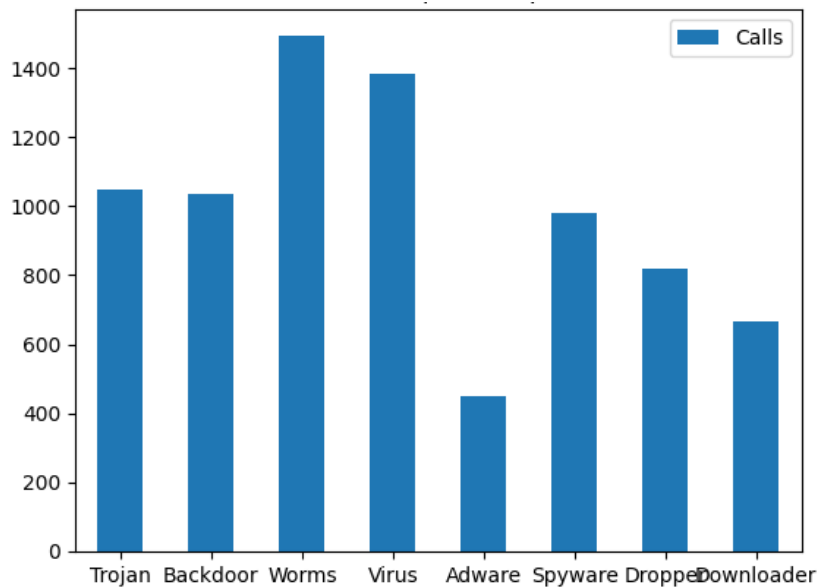


Figure 33: Longueur moyenne d'un sample pour chaque classe

Ceci nous permet de remarquer qu'Adware possède des appels plus courts que la moyenne, Downloader également dans une moindre mesure, et au contraire Worms et Virus des appels plus longs. Ces différences les démarques particulièrement du reste, et

permet d'expliquer les bons résultats obtenus par tous les algorithmes dans leur classification.

Ensuite, nous allons nous intéresser aux nombre d'appels uniques par classe, en même temps que le nombre moyen d'appels uniques par sample par classe.

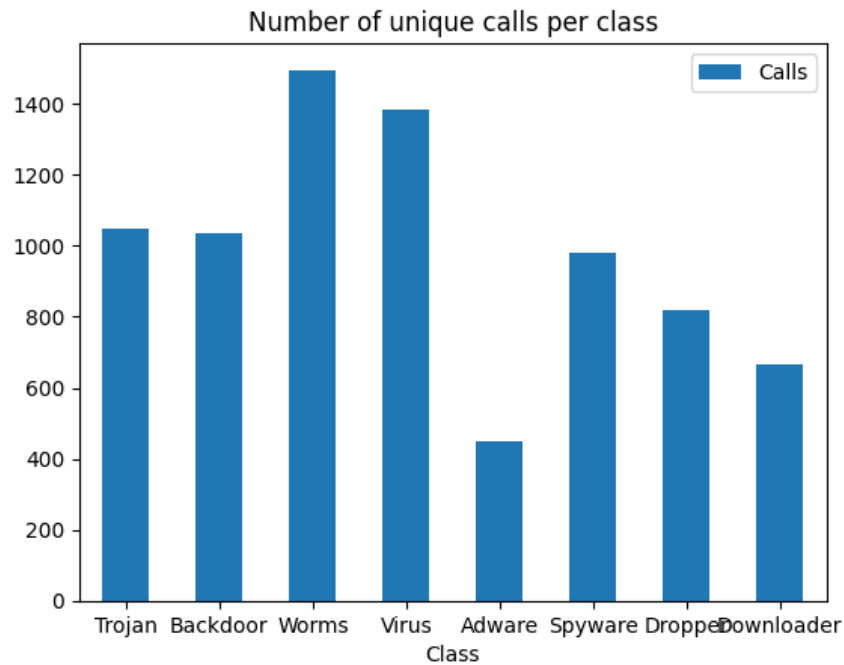


Figure 34: Nombre d'appels uniques par classe

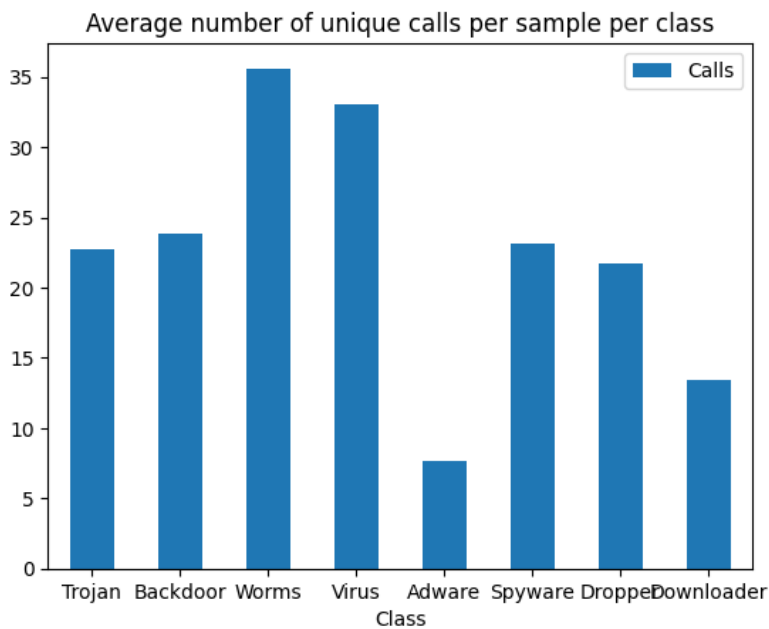


Figure 35: Nombre moyens d'appels api uniques par sample pour chaque classe

Ces deux analyses permettent de pousser les observations précédentes. Les classes Worms, Virus, Downloader et Adware sortent du lot, mais il est difficile de trouver des différences marquées entre les quatres classes réstantes.

Nous allons finalement analyser les appels les plus fréquents de chaque classe, rangé par ordre du plus fréquent au moins fréquent. Les valeurs sont vectorisées afin d'être plus faciles à manipuler. Cela permet d'avoir une vue de tous les appels qui ont été au moins une fois les plus fréquents pour chaque classe.

	Class	most_commons
0	Trojan	{1, 2, 3, 5, 10, 23, 119, 345}
1	Backdoor	{1, 2, 3, 12, 18001}
2	Worms	{1, 2, 3, 4, 201, 173, 1206, 23, 30}
3	Virus	{1, 2, 3, 5, 6, 11, 86321}
4	Adware	{1, 2, 3}
5	Spyware	{1, 2, 3, 100, 6, 8, 10, 11}
6	Dropper	{1, 2, 3, 4, 6, 18001, 150, 152}
7	Downloader	{1, 2, 3, 4, 6, 206, 244, 20}

Figure 36: Appels les plus fréquents par classe, vectorisés

On peut ainsi remarque qu'entre les quatres catégories restantes, Backdoor sort du lot par un nombre plus faibles d'appels les plus fréquents. Cela pourrait expliquer les bons résultats de cette catégorie par rapport aux trois restantes. Cependant, il est difficile de trouver de grande différence entre les trois dernières classes.

En conclusion, pour chaque classe possédant de bon scores d'évaluation sur les différents alorithmes, on a pu s'appercevoir lors d'une des étapes de cette analyse de données un particularité permettant de l'identifier par rapport aux autres. Cependant, pour les trois classes restantes, les samples d'entrées semblent être relativement similaires. Aussi bien dans leur longueur que leur diversité, il nous a été difficile d'identifier un parmètre permettant de mettre en avant une des particularité des ces classes entre elles. Cela pourrait expliquer la difficulté rencontré par tous les modèles pour la classification de ces dernières.

7 Annexes

En plus de ce document, les annexes suivantes sont fournies :

- `notebook_detection.ipynb` : notebook retraçant nos travaux sur la partie detection de malware
- `notebook_classification.ipynb` : notebook retraçant nos travaux sur la partie classification de malware
- `malware_detection.py` : application peremettant de tester et visualiser les résultats des différents modèles dans le cas de la détection

- `malware_classification.py` : application permettant de tester et visualiser les résultats des différents modèles dans le cas de la classification

Les ressources sont trouvables au lien suivant : <https://github.com/NathSimon/SR2I208>

List of Figures

1	Dataset de détection	5
2	Dataset de classifications	6
3	Labels associés	6
4	Vocabulaire de la vectorisation	7
5	Résultat de la vectorisation	7
6	Exemples de courbes ROC pour différents fit	17
7	Exemples d'histogrammes pour différents fit	17
8	Courbe ROC et score AUC pour Random Forest	20
9	Histogramme pour Random Forest	20
10	Courbe ROC et score AUC pour SVM	21
11	histogramme pour SVM	21
12	Courbe ROC et score AUC pour AdaBoost	22
13	histogramme pour AdaBoost	22
14	Courbe ROC et score AUC pour XGBoost	23
15	histogramme pour XGBoost	23
16	Courbe ROC et score AUC pour MLPC	24
17	histogramme pour MLPC	24
18	Courbe ROC et score AUC pour le modèle de Deep Learning	25
19	histogramme pour le modèle de Deep Learning	25
20	Courbes ROC et histogrammes pour Adaboost - partie 1	29
21	Courbes ROC et histogrammes pour Adaboost - partie 2	29
22	Courbes ROC et histogrammes pour Random Forest - partie 1	30
23	Courbes ROC et histogrammes pour Random Forest - partie 2	31
24	Courbes ROC et histogrammes pour XGBoost - partie 1	32
25	Courbes ROC et histogrammes pour XGBoost - partie 2	32
26	Courbes ROC et histogrammes pour SVM - partie 1	33
27	Courbes ROC et histogrammes pour SVM - partie 2	34
28	Courbes ROC et histogrammes pour MLPC - partie 1	35
29	Courbes ROC et histogrammes pour MLPC - partie 2	35

30	Courbes ROC et histogrammes pour DL - partie 1	36
31	Courbes ROC et histogrammes pour DL - partie 2	37
32	Nombre de samples par classe	40
33	Longueur moyenne d'un sample pour chaque classe	40
34	Nombre d'appels uniques par classe	41
35	Nombre moyens d'appels api uniques par sample pour chaque classe . . .	41
36	Appels les plus fréquents par classe, vectorisés	42

List of Tables

1	Matrice de confusion pour une classification sur 3 labels distincts	13
2	Résultats généraux des algorithmes mis en place pour la détection	18
3	Détail des résultats de Random Forest pour la détection	18
4	Détail des résultats de SVM pour la détection	18
5	Détail des résultats de Adaboost pour la détection	18
6	Détail des résultats de XGBoost pour la détection	19
7	Détail des résultats de MLPC pour la détection	19
8	Détail des résultats de DL pour la détection	19
9	Résumé des scores AUC	25
10	Résultats généraux des algorithmes mis en place pour la classification . .	26
11	Détail des résultats de SVM pour la classification	26
12	Détail des résultats de Adaboost pour la classification	27
13	Détail des résultats de XGBoost pour la classification	27
14	Détail des résultats de MLPC pour la classification	27
15	Détail des résultats de Random Forest pour la classification	28
16	Détail des résultats de DL pour la classification	28
17	Scores AUC en OvR pour AdaBoost	30
18	Scores AUC en OvR pour Random Forest	31
19	Scores AUC en OvR pour XGBoost	33
20	Scores AUC en OvR pour SVM	34
21	Scores AUC en OvR pour MLPC	36
22	Scores AUC en OvR pour DL	37

References

- [1] A. Olivera, “Malware Analysis Datasets: API Call Sequences.” <https://www.kaggle.com/datasets/ang3loliveira/malware-analysis-datasets-api-call-sequences?resource=download>.
- [2] A. S. de Oliveira and R. J. Sassi, “Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks,” 11 2019.
- [3] F. O. Catak, J. Ahmed, K. Sahinbas, and Z. H. Khand, “Data augmentation based malware detection using convolutional neural networks,” *PeerJ Computer Science*, vol. 7, p. e346, Jan. 2021.
- [4] “Scikit-learn TrainTestSplit.” https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.
- [5] “Scikit-learn SVM.” <https://scikit-learn.org/stable/modules/svm.html#svm>.
- [6] “Scikit-learn AdaboostClassifier.” <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>.
- [7] “Scikit-learn GradientBoostingClassifier.” <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>.
- [8] “XGBoost Documentation.” <https://xgboost.readthedocs.io/en/stable//>.
- [9] “Scikit-learn MLPC.” https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.
- [10] “Scikit-learn RandomForestClassifier.” <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.
- [11] “Keras Sequential.” https://keras.io/guides/sequential_model/.
- [12] “Towards Data Science deep dive into multi-label classification..! (with detailed case study).” <https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff>.
- [13] “Towards Data Science evaluating multi-label classifiers.” <https://towardsdatascience.com/evaluating-multi-label-classifiers-a31be83da6ea>.
- [14] M. Grandini, E. Bagli, and G. Visani, “Metrics for multi-class classification: an overview,” 2020.
- [15] “sklearn.metrics.roccurve.” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html.
- [16] “Towards Data Science, Multiclass classification evaluation with ROC Curves and ROC AUC.” <https://towardsdatascience.com/multiclass-classification-evaluation-with-roc-curves-and-roc-auc-294fd4617e3a>.

- [17] “sklearn.metrics.roc_auc_score.” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html.
- [18] “ROC Curve, Multiclass.” <https://github.com/vinyluis/Articles/blob/main/ROC%20Curve%20and%20ROC%20AUC/ROC%20Curve%20-%20Multiclass.ipynb>.
- [19] “Towards Data Science micro, macro weighted averages of f1 score, clearly explained.” <https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f>.
- [20] “Scikit-learn countvectorizer.” https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.
- [21] “Scikit-learn multioutput module.” <https://scikit-learn.org/stable/modules/multiclass.html>.
- [22] “Distill a gentle introduction to graph neural networks.” <https://distill.pub/2021/gnn-intro/>.