Automatisation du processus d'entraînement d'un ensemble d'algorithmes de machine learning optimisés pour la détection d'intrusion

Maxime LABONNE[1,2], Alexis OLIVEREAU[1], et Djamal ZEGHLACHE[3]

[1] Institut LIST, CEA, F-91120, Palaiseau, France
[2] Université Paris-Saclay, France
[3] Institut Télécom Télécom SudParis, Évry, France

maxime.labonne@cea.fr

Résumé. Le déploiement à grande échelle de systèmes de détection d'intrusion (IDS – Intrusion Detection System) par anomalies est empêché par le trop grand nombre de faux positifs qu'ils génèrent. Pour diminuer ce nombre tout en améliorant la précision de la détection, il est nécessaire d'adapter au mieux l'IDS au réseau qu'il doit surveiller. Ainsi, dans le cas d'algorithmes de machine learning supervisés, le processus d'apprentissage est la partie critique dont va dépendre la qualité de la détection. La méthode proposée dans ce papier automatise l'optimisation de ce processus. Elle commence par sélectionner les meilleures techniques d'augmentation de données, afin de modifier le dataset d'apprentissage (NSL-KDD ici). Ce jeu de données est transformé pour surreprésenter successivement chacune des catégories d'attaque. Elle entraîne ensuite une série d'algorithmes de machine learning sur chaque dataset modifié. Ces algorithmes se spécialisent ainsi dans la catégorie surreprésentée. Enfin, ils sont combinés de manière optimale dans un ensemble, afin d'obtenir une meilleure précision que celle du meilleur algorithme utilisé séparément. Cette méthode présente deux avantages principaux : elle permet d'obtenir des résultats supérieurs à l'état de l'art (86.59% d'accuracy globale en test sur NSL-KDD), et ne nécessite aucune intervention humaine.

Mots-clefs : protection-défense-détection, détection d'intrusion, IDS, machine learning, réseau de neurones, ensemble learning, NSL-KDD.

1 Introduction

La sécurité des réseaux informatiques est devenue un secteur de plus en plus critique au cours des dernières années, avec l'apparition de nouvelles attaques fortement médiatisées, capables de paralyser les outils de production des entreprises. Si ces attaques ne peuvent être évitées, il est néanmoins possible de chercher à en diminuer l'impact. Dans ce contexte, les Systèmes de Détection d'Intrusion (IDS – Intrusion Detection System) sont un élément essentiel dans une politique de sécurité.

La détection d'intrusion peut être réalisée sur une machine (HIDS – Host-based Intrusion Detection System) ou sur un réseau (NIDS – Network-based Intrusion Detection System). Les IDS sont également catégorisés selon la méthode utilisée : par anomalie ou par signature. Cette dernière est la méthode la plus couramment employée. Popularisée par des logiciels comme Snort [1] ou Suricata [2], elle compare le trafic analysé à une base de données de signatures d'attaques. Cette méthode repose ainsi sur la connaissance d'attaques déjà rencontrées : elle est incapable d'en détecter de nouvelles. Ce n'est pas le cas de la détection d'anomalies, qui permet d'apprendre un trafic réseau pour signaler ensuite tout comportement anormal. Si cette seconde méthode peut détecter des attaques encore jamais rencontrées, elle a tendance à générer une grande quantité de faux positifs ; ce qui limite son utilisation dans des conditions réelles.

Le machine learning est une technique couramment utilisée pour la détection d'anomalies. L'algorithme apprend typiquement de manière supervisée le trafic réseau sur un set d'entraînement, qui peut contenir des attaques. Chaque donnée du dataset est labellisée comme normale ou comme une attaque. Une fois entraîné, cet algorithme est utilisé sur un dataset de test pour évaluer ses performances sur un trafic qu'il n'a encore jamais rencontré. Dans le cas d'un déploiement réel, l'algorithme va créer son propre set d'entraînement en écoutant le trafic réseau pendant un certain temps. Il peut ensuite analyser le trafic en continu comme s'il s'agissait d'un dataset de test.

Ce papier est organisé de la façon suivante : le dataset utilisé dans cette étude et son prétraitement sont présentés dans la section 2. La section 3 décrit la méthode de sélection des algorithmes de machine learning. La section 4 détaille le processus de combinaison de ces algorithmes pour former des ensembles. Enfin, la section 5 conclut en proposant des pistes de recherche.

2 Prétraitement des données

2.1 Prétraitement du dataset

Nous utilisons pour ce travail le dataset de détection d'intrusion NSL-KDD. Il repose sur un autre dataset populaire, le KDD Cup 99. Ce dernier fut créé en 1999 pour une compétition de machine learning [3]. Le but de cette compétition était de classifier correctement des connexions réseau en 5 catégories : normal, déni de service (DoS), sonde réseau (probe), distant à local (R2L – Remote to Local), utilisateur à root (U2R – User to Root). Chaque connexion possède 41 caractéristiques qui permettent au classificateur de prédire correctement sa classe. Ces caractéristiques sont des informations ou des statistiques calculées à partir de l'écoute d'un réseau local simulé de l'U.S. Air Force en 1998 : durée de la connexion, type du protocole, pourcentage de connexions au même service, etc.

NSL-KDD a été créé en 2009 pour résoudre certains problèmes inhérents à KDD Cup 99 [4]. Il reprend ainsi les mêmes données que ce dernier, mais le modifie grandement pour apporter ses corrections. Ainsi, les connexions redondantes ou dupliquées, qui composaient de 75% à 78% du dataset, ont été supprimées. Le nombre total de données s'en trouve grandement réduit : il passe de 805050 connexions pour KDD Cup

99 à 148517 pour NSL-KDD. Le dataset propose également différents niveaux de difficulté, que nous n'utiliserons pas dans ce papier.

KDD Cup 99 a été critiqué à de nombreuses reprises par la communauté scientifique pour les problèmes que nous avons évoqués [4]. Bien que NSL-KDD en corrige une partie, il est fondé sur les mêmes données qui datent de 1998. Ces dernières sont donc bien trop anciennes pour constituer une représentation fiable de l'activité et des menaces d'un réseau actuel. Il est également à noter la forte disparité dans la distribution des classes de NSL-KDD entre set d'entraînement et de test (voir Tableau 1). Cette différence crée une situation réaliste dans la mesure où la distribution du set d'entraînement ne correspond pas parfaitement avec la situation réelle où l'IDS est déployé. La surinterprétation (ou overfitting) des données d'entraînement par l'algorithme de machine learning est donc à proscrire pour obtenir de bons résultats sur le set de test.

Nous utiliserons NSL-KDD dans ce papier afin de pouvoir comparer nos résultats avec la littérature scientifique, qui utilise majoritairement KDD Cup 99 et NSL-KDD. Pour autant, ce travail est transposable à d'autres données, notamment directement issues de fichiers pcap ou du protocole NetFlow.

Tableau 1. Distribution des classes de NSL-KDD

	Normal	***DoS***	***Probe***	***R2L***	***U2R***
Entraînement	67343 (53.46%)	45927 (36.46%)	11656 (9.25%)	995 (0.79%)	52 (0.04%)
Test	9711 (43.08%)	7460 (33.09%)	2421 (10.74%)	2885 (12.80%)	67 (0.30%)

Le processus de prétraitement modifie le dataset pour le rendre lisible par certains algorithmes, comme le réseau de neurones. Nous l'appliquons ainsi aux sets d'entraînement et de test de NSL-KDD. Premièrement, les labels des différentes attaques sont convertis dans les 5 classes que nous cherchons à classifier. Toutes les données catégoriques sont ensuite transformées en données numériques via encodage one-hot [5]. Enfin, les valeurs de toutes les données sont normalisées entre 0 et 1.

2.2 Augmentation des données

En dehors des différences de distribution entre set d'entraînement et de test, le Tableau 1 montre un important déséquilibre entre les classes d'un même dataset. En effet, les connexions normales composent plus de la moitié du set d'entraînement, tandis que les attaques user to root (U2R) représentent moins d'1% du total. L'algorithme sera donc bien davantage entraîné à reconnaître des connexions normales que des attaques U2R, qui sont minoritaires au point que les négliger impactera très faiblement ses performances de détection.

Or, nous souhaitons obtenir un IDS capable de détecter correctement tous les classes. Cette volonté est ici renforcée par la disparité des distributions entre sets d'entraînement

et de test. Pour améliorer les performances de détection de l'IDS, nous avons tout intérêt à l'entraîner au mieux sur toutes les classes, plutôt que d'imiter la distribution du set d'entraînement.

Apprendre d'un jeu de données déséquilibré (imbalanced learning) est un problème classique de machine learning [6], [7]. La première étape dans notre processus d'optimisation est ainsi de rééquilibrer les données d'entraînement. Il y a deux façons de rééquilibrer des classes : le sous-échantillonnage et le sur-échantillonnage. Comme les noms l'indiquent, le sous-échantillonnage réduit les populations des classes les plus représentées, alors que le sur-échantillonnage augmente celles des classes les moins représentées. Dans notre cas, nous voulons créer 5 datasets d'entraînement : un spécialisé dans chaque classe. Ainsi, la classe spécialisée est sur-échantillonnée, tandis que les autres sont sous-échantillonnées avec un ratio de 1:10. Le classificateur qui apprendra sur ces datasets spécialisés deviendra lui-même spécialiste de la classe la plus représentée.

Deux algorithmes sont donc nécessaires pour chaque classe : un pour le sous-échantillonnage, et un autre pour le sur-échantillonnage. Le meilleur algorithme de sous-échantillonnage est combiné avec le meilleur algorithme de sur-échantillonnage pour former un dataset spécialisé sur une classe avec un ratio de 1:10. Un classificateur naïf est utilisé afin de déterminer les performances de chaque algorithme pour chaque classe. Il s'agit d'un perceptron multicouche (ou MLP – Multilayer Perceptron) composé d'une seule couche cachée de 128 neurones. Il utilise le Rectifier Linear Unit (ReLU) comme fonction d'activation [8], et l'optimiseur Adam [9]. L'aire sous la courbe ROC (Receiver Operating Characteristic) de la classe spécialisée sert de métrique pour mesurer la performance de notre classificateur. Cette métrique correspond à la probabilité pour qu'un exemple positif aléatoire soit classé au-dessus d'un exemple négatif aléatoire [10]. On entraîne ce réseau de neurones sur 80% du set d'entraînement, et on mesure l'aire sous la courbe ROC sur les 20% restants (voir résultats Tableau 2).

Tableau 2. Comparaison d'algorithmes d'augmentation de données pour NSL-KDD

Method	**AUC ROC**				
	Normal	***DoS***	***Probe***	***R2L***	***U2R***
Aucun échantillonnage	0.9488	0.8930	0.9158	0.7429	0.9605
Sous-échantillonnage					
Cluster Centroids	0.9378	0.8915	0.9181	0.8548	**0.9739**
Random Under Sampler	0.9616	0.9000	0.9338	**0.8674**	0.9715
NearMiss	0.9377	0.7491	0.8691	0.5929	0.2762
Edited Nearest Neighbours	0.9382	0.9071	0.9234	0.7932	0.9575
Repeated Edited Nearest Neighbours	0.9614	**0.9162**	0.9081	0.7773	0.9626
Condensed Nearest Neigbour	0.9022	0.6521	0.8518	0.6224	0.9174
One Sided Selection	0.9415	0.6498	**0.9543**	0.6986	0.9435
AllKNN	**0.9659**	0.9019	0.9138	0.8012	0.9701

Method	AUC ROC				
	Normal	***DoS***	***Probe***	***R2L***	***U2R***
Instance Hardness Threshold	0.6944	0.9079	0.8753	0.8104	0.9458
Sur-échantillonnage					
Random Over Sampling	0.9611	0.9083	**0.9364**	0.7838	0.9714
SMOTE	0.9423	0.8965	0.8918	0.8025	0.9511
Borderline 1 SMOTE	0.9597	0.9133	0.9173	**0.8535**	0.9675
Borderline 2 SMOTE	0.9542	0.9146	0.9125	0.7659	0.9601
SVM SMOTE	**0.9644**	0.8924	0.9273	0.7990	0.9696
ADASYN	0.9485	**0.9188**	0.9324	0.7855	0.9717
SMOTEENN	0.9500	0.8742	0.9457	0.7365	0.9560
SMOTE Tomek	0.9627	0.8784	0.9178	0.7420	**0.9799**

Le Tableau 2 montre les meilleures méthodes de sous-échantillonnage et de sur-échantillonnage pour chaque classe. Ces deux méthodes sont combinées comme décrit précédemment pour former le set d'entraînement de la classe.

3 Entraînement des classificateurs

Nous utilisons dans cette section différents algorithmes de machine learning pour de la classification : decision tree, random forest, extra tree, extra trees, k-Nearest Neighbors (k-NN), SVM, régression logistique, Naive Bayes, gradient boosting et multilayer perceptron (MLP). Ces classificateurs sont issus de la bibliothèque sklearn [11], à l'exception du MLP créé avec Keras [12] et Tensorflow [13].

Chaque algorithme possède un ensemble de paramètres dont les valeurs influent sur les performances de classification. L'optimisation de ces paramètres est donc une étape importante dans l'amélioration de la précision de la détection. Plusieurs méthodes d'optimisation sont couramment utilisées en machine learning. La plus courante fait appel à un expert qui entre les paramètres les plus pertinents pour le problème étudié. Cette méthode demande une bonne connaissance de l'algorithme à optimiser ainsi que l'impact des différents paramètres. Il serait possible de fixer ici les meilleurs paramètres pour le dataset étudié (NSL-KDD). Mais dans l'optique de proposer une méthode transposable à d'autres données, nous souhaitons automatiser la recherche des paramètres.

En définissant un ensemble de recherche pour chaque paramètre à optimiser, il est possible de tester toutes les combinaisons de valeurs de paramètres possibles. Cette technique, appelée grid search, a l'avantage de trouver nécessairement l'optimum global, c'est-à-dire la combinaison de paramètres qui offre les meilleurs résultats de classification. Elle est cependant difficilement applicable en pratique, du fait du au nombre important de paramètres définis sur des ensembles de recherche étendus. Une variante, le random search, consiste à tirer aléatoirement un nombre défini de combinaisons à

tester. Cette technique permet de gérer précisément le temps alloué au processus d'optimisation, et a l'avantage de converger plus rapidement que le grid search [14].

Nous utilisons dans ce papier une autre technique : le Tree-structured Parzen Estimator (TPE). Le TPE est un algorithme de type Sequential Model-Based Optimization (SMBO) [15]. Contrairement au random search, il sélectionne intelligemment le prochain ensemble de paramètres à tester, en connaissant les performances obtenues par les précédents. Des tests réalisés durant ce travail ont permis de montrer que le TPE convergeait non seulement plus vite que le random search, mais obtenait également de meilleures performances de classification (un meilleur optimum) sur ce problème.

La métrique utilisée pour représenter la performance de classification est à nouveau l'aire sous la courbe. Chaque algorithme de machine learning est entraîné sur 80% du set d'entraînement de chaque classe, obtenu dans la section précédente. Ses performances sont ensuite évaluées sur les 20% restants, issus de set d'entraînement original de NSL-KDD (donc sans l'échantillonnage spécifique à la classe). Le but de TPE est de trouver, pour chaque classificateur, les paramètres qui maximisent l'aire sous la courbe calculée sur ces 20%. Ces paramètres sont sauvegardés pour chaque algorithme appliqué au set d'entraînement de chaque classe, et seront réutilisés dans la suite du papier.

4 Ensemble d'algorithmes

Les modèles entraînés sont combinés pour former un ensemble, dont la précision est meilleure que celle du meilleur élément qui le compose. Cette approche est couramment utilisée avec succès dans les compétitions de machine learning, à l'image de KDD Cup 2009 [10]. Elle repose sur l'idée qu'une combinaison de classificateurs faibles vaut mieux qu'un seul classificateur fort. La règle de combinaison choisie a donc une importance majeure sur la qualité des résultats.

Nous souhaitons créer un ensemble pour chaque classe, en combinant les meilleurs classificateurs de cette classe. En effet, tous les modèles entraînés n'ont pas les mêmes performances de détection. Ainsi, ajouter des modèles peu précis entraîne une accumulation des erreurs, davantage que des détections correctes. Une sélection des meilleurs modèles permet d'empêcher cette dégradation de la qualité de la détection.

D'autre part, notre volonté de procéder classe par classe est motivée par les spécificités des algorithmes par rapport aux données qu'on leur apporte en entrée. Une façon d'appréhender cette problématique est de la voir comme un compromis entre biais et variance. Certains classificateurs sont plus efficaces sur des données avec un fort biais et une faible variance (Naive Bayes par exemple), alors que d'autres sont plus performants pour un faible biais et une forte variance (k-NN par exemple). Pour de petits datasets, on préférera généralement les algorithmes de la première catégorie, alors que les autres seront plus adaptés pour de larges datasets.

La sélection des classificateurs est souvent effectuée manuellement, ou en corrélant les résultats de chaque algorithme pour ne retenir que les moins corrélés. Une méthode automatique novatrice est adoptée ici, permettant d'attribuer avec souplesse un coefficient de pondération aux prédictions de chaque algorithme de l'ensemble. Les modèles

sont tout d'abord classés du meilleur au moins bon, d'après leur aire sous la courbe de ROC de leur classe sur le set de validation. Les prédictions p_1 du moins bon modèle et celles p_2 du second moins bon modèle sont combinées par une moyenne pondérée (équation 1). Un TPE tente alors d'optimiser les ratios λ_1 et λ_2 de la moyenne afin d'obtenir la meilleure précision (au sens de l'accuracy) possible sur ce set. Ces ratios sont conservés et le processus se répète entre l'ensemble formé, et le nouvel algorithme qu'on tente d'ajouter. Si aucun ratio ne permet d'améliorer la précision de l'ensemble, le nouvel algorithme est ignoré et le suivant est testé à sa place. Le choix de commencer par les moins bons modèles s'explique par la diminution progressive du ratio des premiers modèles, au fur et à mesure que de nouveaux sont ajoutés.

$$p = \sum_{i=1}^{N} \lambda_i p_i \tag{1}$$

Cette méthode aboutit à la création de 5 ensembles, chacun spécialisé dans une classe de NSL-KDD. Chaque ensemble produit une prédiction p, qui est la somme pondérée des prédictions p_i de chaque algorithme i par un ratio λ_i L'overfitting est atténué par l'ajout des moins bons modèles au début du processus, qui contribuent néanmoins à la classification.

Ces ensembles sont ensuite combinés à leur tour pour obtenir la classification finale sur le set de test. Nous retenons l'ensemble qui attribue la probabilité la plus haute à sa classe pour déterminer la classe finale de chaque connexion. Cette règle de combinaison repose sur l'idée qu'une prédiction erronée aura la plupart du temps une probabilité moins élevée qu'une prédiction correcte. Nous obtenons avec cette méthode une accuracy globale (indice de Jaccard) sur les 5 classes du set de test de NSL-KDD de 86.59%. Il s'agit, à notre connaissance, de l'accuracy la plus élevée obtenue dans la littérature (la valeur maximale observée jusqu'alors étant de 82.68% avec un Support Vector Machine [17]). Le taux de faux positifs (trafic normal classifié comme une attaque) reste également faible, avec 1.66%.

Nous présentons ci-dessous les métriques utilisées (voir Tableau 3 et équations 2, 3, 4, 5 et 6) pour le tableau récapitulatif des résultats obtenus.

Tableau 3. Matrice de confusion

		Résultat prédit	
		Négatif	***Positif***
Résultat réel	***Négatif***	Vrai Négatif (VN)	Faux Positif (FP)
	Positif	Faux Négatif (FN)	Vrai Positif (VP)

- L'accuracy est le ratio des résultats correctement identifiés.

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN} \quad (2)$$

- Le Taux de Vrais Positifs (TVP) est la proportion de positifs qui sont correctement détectés.

$$TVP = \frac{VP}{VP + FN} \quad (3)$$

- Le Taux de Faux Positifs (TFP) est la proportion de négatifs incorrectement détectés comme des positifs.

$$TFP = \frac{FP}{VN + FP} \quad (4)$$

- La précision est la proportion de prédictions de positifs qui sont en effet des positifs.

$$Précision = \frac{VP}{VP + FP} \quad (5)$$

- Le F1 score est la moyenne harmonique de la précision et du TVP.

$$F1\ score = 2 \times \frac{précision \times TVP}{précision + TVP} \quad (6)$$

L'ensemble des résultats obtenus pour chaque classe est détaillé dans le Tableau 4 ci-dessous. L'AUC ROC correspond à l'aire sous la courbe de ROC.

Tableau 4. Taleau des résultats obtenus pour la classification du set de test de NSL-KDD

	Normal	***DoS***	***Probe***	***R2L***	***U2R***
Accuracy	95.20%	90.91%	95.87%	91.73%	99.48%
TVP	88.85%	95.99%	86.70%	50.02%	43.28%
TFP	1.66%	12.94%	3.03%	2.15%	0.36%
F1 score	0.9245	0.9009	0.8185	0.6076	0.3295
AUC ROC	0.9359	0.9153	0.9184	0.7394	0.7146

5 Conclusion

Nous avons présenté dans ce papier une méthode d'optimisation automatique d'un ensemble d'algorithmes de machine learning pour la détection d'intrusion. En premier lieu, le set d'entraînement est prétraité dans le but de générer un dataset spécialisé pour

chaque classe. Une série d'algorithmes de machine learning est entraînée (et, dans certains cas, optimisée) pour chacun de ces datasets. Enfin, les prédictions des meilleurs algorithmes sont combinées en optimisant leurs pondérations pour améliorer la précision de la détection. Cette méthode offre deux avantages principaux : premièrement, ses résultats sont supérieurs à l'état de l'art avec une précision de 86.59% sur NSL-KDD. Dans un second temps, elle permet d'envisager un déploiement automatique d'une solution d'IDS à base de machine learning.

NSL-KDD ne constitue néanmoins pas un vrai trafic réseau. Cette méthode repose en effet sur un apprentissage supervisé, et nécessite ainsi un set d'entraînement labellisé. Pour un vrai déploiement, elle devrait reposer sur un jeu de données pré-établi, ou une méthode de labellisation automatique. Notre processus pourrait également être amélioré en gérant une sélection automatique des caractéristiques en entrée algorithme par algorithme. Cela permettrait de supprimer les paramètres les moins importants du dataset pour en diminuer la dimensionnalité. Cette réduction raccourcirait le temps de calcul, et augmenterait la précision de la détection, particulièrement pour les algorithmes de machine learning fortement soumis au fléau de la dimension.

6 Références

[1] « Snort - Network Intrusion Detection & Prevention System ». [En ligne]. Disponible sur: https://www.snort.org/. [Consulté le: 28-juin-2017].

[2] « Suricata », *Suricata*. [En ligne]. Disponible sur: https://suricata-ids.org/. [Consulté le: 28-juin-2017].

[3] « KDD-CUP-99 Task Description ». [En ligne]. Disponible sur: https://kdd.ics.uci.edu/databases/kddcup99/task.html. [Consulté le: 18-juin-2017].

[4] M. Tavallaee, E. Bagheri, W. Lu, et A. A. Ghorbani, « A detailed analysis of the KDD CUP 99 data set », in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, 2009.

[5] M. Z. Alaya, S. Bussy, S. Gaïffas, et A. Guilloux, « Binarsity: a penalization for one-hot encoded features », *ArXiv170308619 Stat*, mars 2017.

[6] H. He et E. A. Garcia, « Learning from Imbalanced Data », *IEEE Trans. Knowl. Data Eng.*, vol. 21, nº 9, p. 1263-1284, sept. 2009.

[7] B. Krawczyk, « Learning from imbalanced data: open challenges and future directions », *Prog. Artif. Intell.*, vol. 5, nº 4, p. 221-232, nov. 2016.

[8] A. Krizhevsky, I. Sutskever, et G. E. Hinton, « Imagenet classification with deep convolutional neural networks », in *Advances in neural information processing systems*, 2012, p. 1097–1105.

[9] D. P. Kingma et J. Ba, « Adam: A Method for Stochastic Optimization », *ArXiv14126980 Cs*, déc. 2014.

[10] A. P. Bradley, « The use of the area under the ROC curve in the evaluation of machine learning algorithms », *Pattern Recognit.*, vol. 30, nº 7, p. 1145-1159, juill. 1997.

[11] F. Pedregosa *et al.*, « Scikit-learn: Machine Learning in Python », *Mach. Learn. PYTHON*, p. 6.

[12] F. Chollet, *Keras*. 2015.
[13] M. Abadi *et al.*, « TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems », *ArXiv160304467 Cs*, mars 2016.
[14] J. Bergstra et Y. Bengio, « Random search for hyper-parameter optimization », *J. Mach. Learn. Res.*, vol. 13, nº Feb, p. 281–305, 2012.
[15] J. S. Bergstra, R. Bardenet, Y. Bengio, et B. Kégl, « Algorithms for hyper-parameter optimization », in *Advances in neural information processing systems*, 2011, p. 2546–2554.
[16] « SIGKDD : KDD Cup 2009 : Customer relationship prediction ». [En ligne]. Disponible sur: http://www.kdd.org/kdd-cup/view/kdd-cup-2009. [Consulté le: 15-avr-2018].
[17] M. S. Pervez et D. M. Farid, « Feature selection and intrusion classification in NSL-KDD cup 99 dataset employing SVMs », in *The 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014)*, 2014, p. 1-6.