Chapitre 2 : Méthodes de l'IA pour les NIDS

Dr. ZAMOUCHE Djamila

Université A. MIRA - Bejaia

Faculté des Science Exactes

Département d'Informatique

Email : djamila.zamouche@univ-bejaia.dz

2024 / 2025

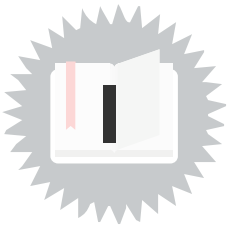
Table des matières

Objectifs

A l'issu de ce chapitre, l'étudiant sera capable de :

- Connaître les techniques Machine Learning (ML) ;
- Utiliser les données pour assurer la sécurité ;
- Utiliser les algorithmes ML pour la détection d'intrusion réseau ;
- Maîtriser les exigences des systèmes de sécurité basés sur ML.

Introduction



L'immense évolution des technologies au cours de la dernière décennie a entraîné une forte expansion de la taille des réseaux et du nombre d'applications. En conséquence, une énorme quantité de données importantes est générée et partagée dans le réseau. La sécurité de ces données est devenue une tâche difficile en raison de la génération d'un grand nombre de nouvelles attaques. Les IDS existants se sont révélés inefficaces pour détecter diverses attaques, et pour réduire le taux de fausses alarmes (FAR). Il en résulte finalement une demande pour un NIDS efficace, précis et rentable afin de fournir une sécurité forte au réseau.

Les chercheurs ont exploré la possibilité d'utiliser des techniques d'apprentissage automatique (Machine Learning (ML)) et d'apprentissage profond (Deep Learning (DL)), qui font partie de l'Intelligence Artificielle (IA). ML et DL sont des outils puissants pour apprendre des caractéristiques utiles du trafic réseau et prédire les activités normales et anormales sur la base des modèles appris.

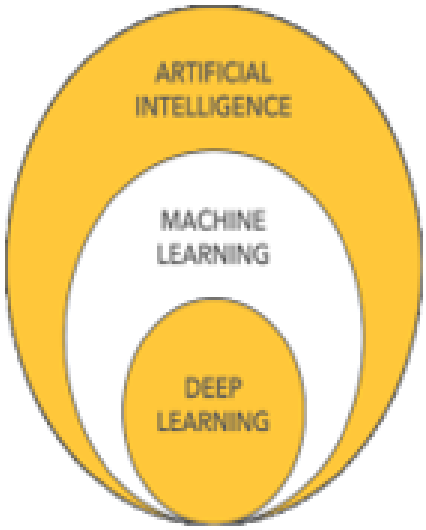
Dans ce chapitre, nous présentons et discuter des applications des techniques de ML dans de nombreux domaines posant des problèmes en matière de sécurité. Nous donnons des exemples sur la façon dont l'apprentissage automatique peut être appliqué à des problèmes comme la détection des attaques réseaux, la classification des logiciels malveillants ou l'analyse réseau. Nous vous fournirons une liste d'algorithmes parmi lesquels vous pourrez faire un choix lorsque vous aborderez un problème de sécurité donné. Nous explorerons également des méthodes permettant d'évaluer l'adéquation de diverses techniques d'apprentissage automatique dans différents scénarios, et nous nous concentrerons sur les principes directeurs qui vous aideront à utiliser les données afin d'obtenir une meilleure sécurité.

Notre but n'est pas de vous fournir les réponses à tous les problèmes de sécurité auxquels vous pourriez être confrontés, mais plutôt de vous donner un cadre de réflexion quant aux données et à la sécurité, ainsi qu'une boîte à outils dans laquelle vous pouvez choisir la bonne méthode en fonction du problème à résoudre.

L'IA pour les NIDS

1. Intelligence Artificielle

- **Intelligence Artificielle (IA) :** est l'ensemble des techniques et théories qui cherchent à développer des modèles capables de simuler le comportement humain.
- **Machine Learning (ML) :** est un sous-ensemble de l'IA qui comprend toutes les méthodes et tous les algorithmes qui permettent aux machines d'apprendre automatiquement à l'aide de modèles mathématiques. ML consiste à utiliser des données passées, provenant d'historiques, pour créer un algorithme de prédiction pour les données futures.
- **Le Deep Learning :** est un domaine du Machine Learning qui est focalisé sur le développement des réseaux de neurones et qui fait face à d'autres défis que ceux du machine Learning. Parmi ces défis, comment entraîner des modèles avec des millions de paramètres et des milliards de données dans des temps raisonnables.



2. Techniques de Machine Learning

Pour donner à un ordinateur la capacité d'apprendre, on utilise des *méthodes d'apprentissage* qui sont fortement inspirées de la façon dont nous, les êtres humains, apprenons à faire des choses. L'apprentissage automatique peut être *supervisé* (Supervised Learning) ou *non supervisé* (Unsupervised Learning).

L'apprentissage automatique supervisé *Définition*

L'apprentissage automatique peut être supervisé dans le cas où des étiquettes sont associées à des données d'apprentissage, et vous essayez de prédire des étiquettes pour des données futures. Par exemple, étant donné un large nombre de courriels étiquetés comme étant du spam, on peut entraîner un classifieur de spams qui essaie de prédire si un nouveau message entrant est du spam. Avec le Supervised Learning on peut développer des modèles pour résoudre 2 types de problèmes :

1. **Les problèmes de Régression** : on cherche à prédire la valeur d'une variable continue, c'est-à-dire une variable qui peut prendre une infinité de valeurs.

   Par exemple, prédire le prix d'un appartement (y) selon sa surface habitable (x) ; Prédire la quantité d'essence consommée (y) selon la distance parcourue (x).

2. **Les problèmes de Classification** : on cherche à classer un objet dans différentes classes, c'est-à-dire que l'on cherche à prédire la valeur d'une variable discrète (qui ne prend qu'un nombre fini de valeurs).

   Par exemple,

   - Prédire si un émail est un spam (classe y = 1) ou non (classe y = 0) selon le nombre de liens présent dans l'émail (x) ;
   - Prédire si une tumeur est maligne (y = 1) ou bénigne (y = 0) selon la taille de la tumeur (x1) et l'âge du patient (x2).

L'apprentissage automatique non supervisé *Définition*

L'apprentissage automatique peut être non supervisé dans le cas où des étiquettes ne sont pas associées à des données d'historique. On peut ne même pas savoir ce que sont les étiquettes que vous essayez de prédire. Le clustering, la détection d'anomalie, etc. sont une forme typique d'apprentissage non supervisé.

Exemple

Vous avez un nombre inconnu de botnets attaquant votre réseau et que vous voulez les distinguer les uns des autres.

Exemple d'algorithmes supervisés et algorithmes non supervisés

Algorithmes supervisés	Algorithmes non supervisé
SVM	K-means
KNN	GMM
Arbres de décision	
Méthodes ensemble	

2.1. Arbre de décision

Définition

Arbre de décision (en anglais Decision Tree (DT)) est l'un des algorithmes ML supervisés de base qui est utilisé pour la classification et la régression d'un ensemble de données donné en appliquant une série de décisions (règles). Le modèle a une structure arborescente avec des nœuds, des branches et des feuilles. Chaque nœud représente un attribut ou une caractéristique. La branche représente une décision ou une règle, tandis que chaque feuille représente un résultat possible ou une étiquette de classe.

L'algorithme DT sélectionne automatiquement les meilleures caractéristiques pour construire un arbre et effectue ensuite une opération d'élagage pour supprimer les branches non pertinentes de l'arbre afin d'éviter le sur-ajustement.

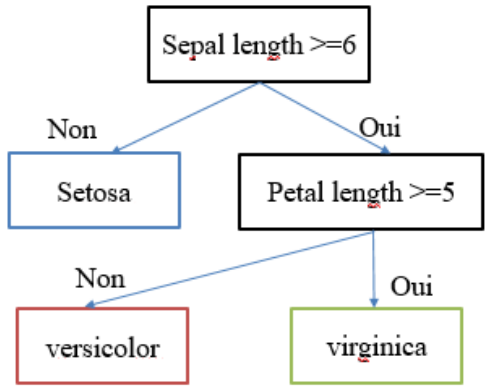
Modèle d'arbre de décision

Pour comprendre intuitivement la stratégie adoptée par les arbres de décision, voyons les étapes typiques de leur mise en œuvre :

- La première étape consiste à subdiviser l'ensemble de données original en deux sous-ensembles, suite à la première subdivision, nous aurons comme résultat deux sous-ensembles de données .
- Les sous-ensembles seront à nouveau subdivisés (d'une manière itérative) sur la base d'autres conditions ; à chaque étape, la condition qui fournit la meilleure bipartition du sous-ensemble original est choisie (à cette fin, des métriques appropriées sont utilisées pour mesurer la qualité de la subdivision).
- La division se déroule de manière itérative. Il est donc nécessaire de définir une condition d'arrêt (telle que l'atteinte d'une profondeur maximale).
- A chaque itération, l'algorithme génère une structure arborescente dont les nœuds représentent les choix faits à chaque fois, chaque feuille contribuant à la classification globale des données d'entrée.

Exemple

Sepal length	Sepal width	Petal length	Petal width	Espèce
5.1	3.5	1.4	0.2	I. setosa
7.0	3.2	4.7	1.4	*I. versicolor*
6.3	3.3	6.0	2.5	*I. virginica*
⋮	⋮	⋮	⋮	⋮



Estimation de probabilité d'appartenance

Complément

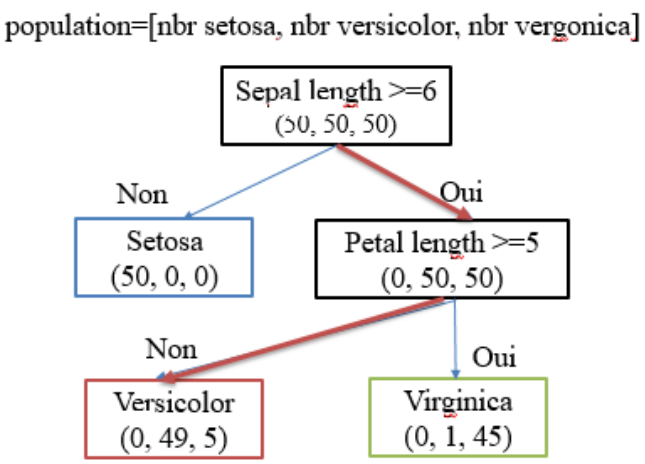
Un arbre de décision peut également estimer la probabilité qu'une observation appartienne à une classe k particulière.

Par exemple : nous voulons identifier une fleur avec un sépale de 6 cm et un pétale de 4cm.

On va voir combien de setosa ont été mis dans cette classe, ici 0. finalement cette observation à 0% de chance d'être setosa.

On va prendre maintenant le nombre de versocolor qui ont été mis dans cette classe : 49 iris versocolor contre 54 de population dans cette classe. Donc, il y a 90,7% de chance que notre nouvelle fleur soit versicolor.

Enfin on avait mal classifié 5 verginica dans notre classe lors de notre entraînement, donc l'observation à 9% de chance d'être vergicica.



2.2. K-plus proches voisins

K-plus proches voisins (Nearest Neighbor (KNN)) est l'un des algorithmes supervisés de ML les plus simples qui utilise l'idée de "similarité des caractéristiques" pour prédire la classe d'un certain échantillon de données. Il identifie un échantillon sur la base de ses voisins en calculant sa distance par rapport à ces derniers.

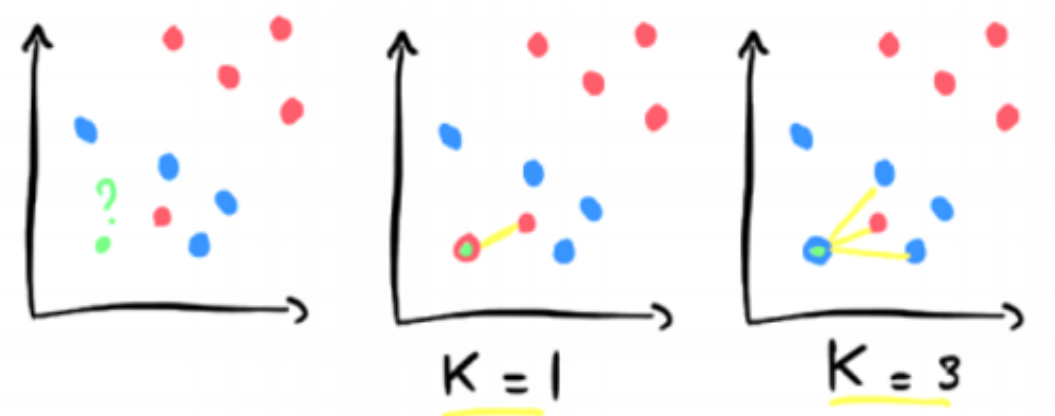
La **phase d'apprentissage** consiste simplement à **stocker** tous les vecteurs de caractéristiques et les étiquettes d'échantillons correspondants dans le modèle. La prédiction de la classification est simplement l'étiquette la plus courante parmi les k voisins les plus proches de l'échantillon de test (d'où le nom). Typiquement, nous utilisons la distance euclidienne (c'est la droite direct entre deux points) mais d'autres méthodes, comme la distance de Manhattan ou bien la distance cosinus.

Le modèle kNN

- **Transformation des données :** convertir les données en un vector space model.
- **Entraînement du modèle :** utiliser l'ensemble de données d'entraînement pour entraîner le kNN. Le modèle stocke les vecteurs de caractéristiques et les étiquettes d'échantillons correspondants.
- **Calcul de la distance :** calcule la distance entre l'échantillon de test x et tous les autres échantillons.
- Trier les distances calculées dans l'ordre croissant.
- **Classification :** sélectionner les k instances les plus proches de x et choisir la classe major.

Exemple

Quel est l'exemple le plus proche du point vert ?



2.3. Machine à vecteur de support

Machine à vecteur de support (Support Vector Machine (SVM)) est une méthode de classification supervisé qui fonctionne en trouvant un hyperplan de séparation (frontière de décision) dans un espace de caractéristiques. Elle est utilisée pour la résolution de problèmes linéaires et non linéaires. L'objectif est de trouver un hyperplan qui sépare au mieux les deux classes, de manière à maximiser la marge entre les deux classes.

L'hyperplan est défini par une équation linéaire : ax + b = 0, où a est le vecteur de poids (qui définit la direction de l'hyperplan) et b est le biais (qui détermine la position de l'hyperplan dans l'espace).

Processus d'entraînement d'un SVM

- **Transformation des données :** transformer les données en un format approprié, tel que des vecteurs de caractéristiques numériques ;
- **Entraînement du modèle :** utiliser l'ensemble de données d'entraînement pour entraîner le SVM. Le modèle trouve l'hyperplan qui maximise la marge tout en maintenant la classification correcte des exemples d'entraînement ;
- **Recherche des vecteurs de support :** les exemples d'entraînement qui se trouvent à proximité de l'hyperplan sont appelés vecteurs de support. Ils sont cruciaux pour la définition de l'hyperplan.
- **Calcul de l'hyperplan :** l'hyperplan est calculé à l'aide des vecteurs de support. Les poids a et le biais b sont déterminés pour définir l'hyperplan de manière optimale ;
- **Évaluation du modèle :** évaluer la performance du modèle SVM à l'aide d'un ensemble de données de test distinct.

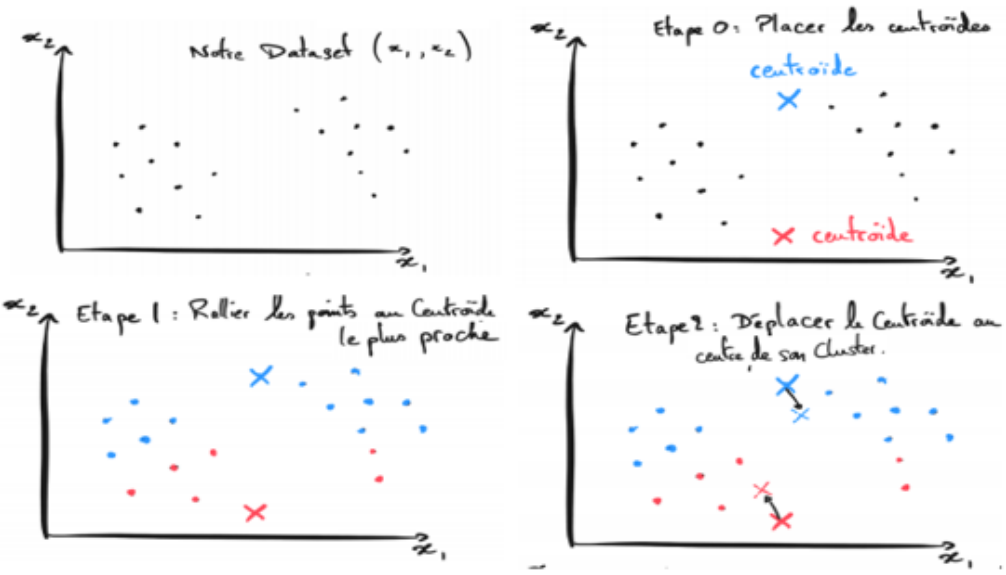
2.4. K-Means

Le K-Mean clustering est sans doute l'algorithme le plus populaire pour les problèmes de clustering, en plaçant les données hautement similaires dans le même cluster. L'algorithme K-Mean est l'un des algorithmes de ML itératifs populaires basés sur les centroïdes qui apprennent de manière non supervisée. K représente le nombre de centroïdes (centre du cluster) dans un ensemble de données. L'algorithme fonctionne en 2 étapes répétées en boucle : On commence par placer au hasard un nombre K de centres dans notre nuage de points. Ensuite, l'étape 1 consiste à rallier chaque exemple au centre le plus proche. L'étape 2 consiste à déplacer les centres au milieu de leur cluster. On répète ainsi les étapes 1 et 2 en boucle jusqu'à ce que les centres ne bougent plus.

Modèle k-means

1. **Choix du nombre de clusters (K) :** déterminer combien de clusters (K) à identifier dans nos données.
2. **Initialisation des centres de cluster :** sélectionner K points initiaux comme centres de cluster. Vous pouvez les choisir aléatoirement ou utiliser une méthode de sélection plus avancée.
3. **Affectation des paquets aux clusters :** calculer la distance entre chaque paquet et les centres de cluster. Affecter chaque paquet au cluster dont le centre est le plus proche en fonction de la distance calculée.
4. **Mise à jour des centres de cluster :** calculer les nouveaux centres de cluster en utilisant la moyenne des paquets attribués à chaque cluster.
5. **Répétition des étapes 3 et 4 :** Répéter les étapes 4 et 5 jusqu'à ce qu'un critère d'arrêt soit atteint (par exemple, lorsque les centres de cluster ne changent que très peu entre les itérations).
6. **Interprétation des clusters :** Une fois que K-means a convergé, examinez les clusters formés pour comprendre leur signification. Chaque cluster peut représenter un type de trafic réseau similaire.

Exemple



2.5. Méthodes d'ensemble

L'idée principale derrière les méthodes d'ensemble (Ensemble Learning) est de tirer profit des différents classifieurs en apprenant de manière groupée. Comme chaque classifieur a des forces et des faiblesses, et certains peuvent être performants pour détecter un type spécifique d'attaques et montrer de mauvaises performances sur d'autres types d'attaques, l'approche d'ensemble consiste à combiner les résultats de plusieurs modèles faibles pour entraîner un seul apprenant plus fort. Le regroupement de modèles pour obtenir les meilleures performances peut se faire de plusieurs façons, en utilisant plusieurs types d'algorithmes de regroupement, à savoir :

1. **Bagging :** C'est de créer plusieurs copies d'un même modèle (par exemple plusieurs arbres de décision), en entraînant chaque modèle sur une partie aléatoire du dataset. Pour ceci, on utilise une technique d'échantillonnage appelée Bootstrapping, et qui consiste à replacer après chaque tirage au sort les données qui ont été sélectionnées dans notre dataset. De cette manière, on obtient un ensemble de modèles diversifiés, parce que ils n'ont pas tous été nourris avec les mêmes données, mais qui partagent certaines connaissances en commun.

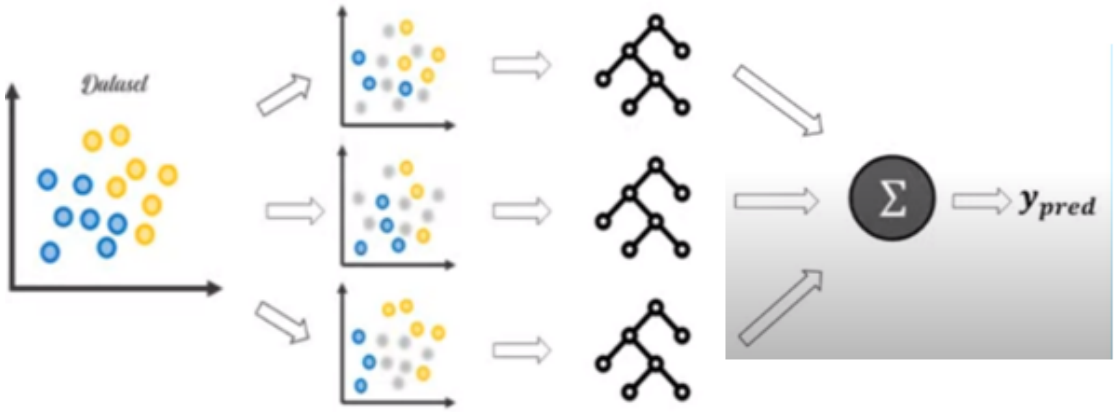
   Les prédictions des modèles individuels sont ensuite agrégées, généralement par un vote majoritaire (pour la classification) ou une moyenne (pour la régression), pour obtenir la prédiction finale.

2. **Boosting :** Cette fois ci, l'idée est d'entraîner l'un après l'autre plusieurs modèles relativement faibles, en demandant à chaque modèle d'essayer de corriger les erreurs effectuées par son prédécesseur. Du coup, on obtient un ensemble de modèles parfaitement complémentaires, dans lequel les faiblesses des uns sont compensées par les forces des autres.

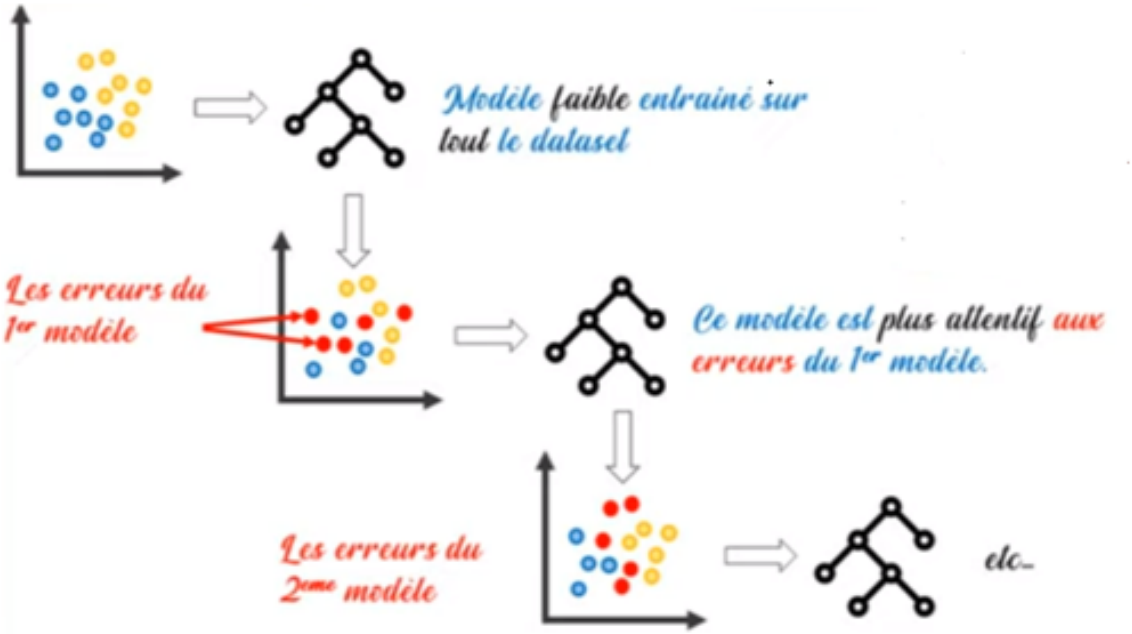
   Les algorithmes de Boosting populaires incluent AdaBoost, Gradient Boosting, XGBoost et LightGBM.

3. **Stacking :** L'idée c'est d'entraîner un modèle de ML par-dessus les prédictions de notre ensemble, c'est-à-dire, au lieu de simplement ressembler les résultats de nos modèles pour retenir une prédiction majoritaire, on demande à un dernier estimateur d'apprendre à reconnaître qui a tort et qui a raison dans notre ensemble pour lui-même prédire le résultat final.

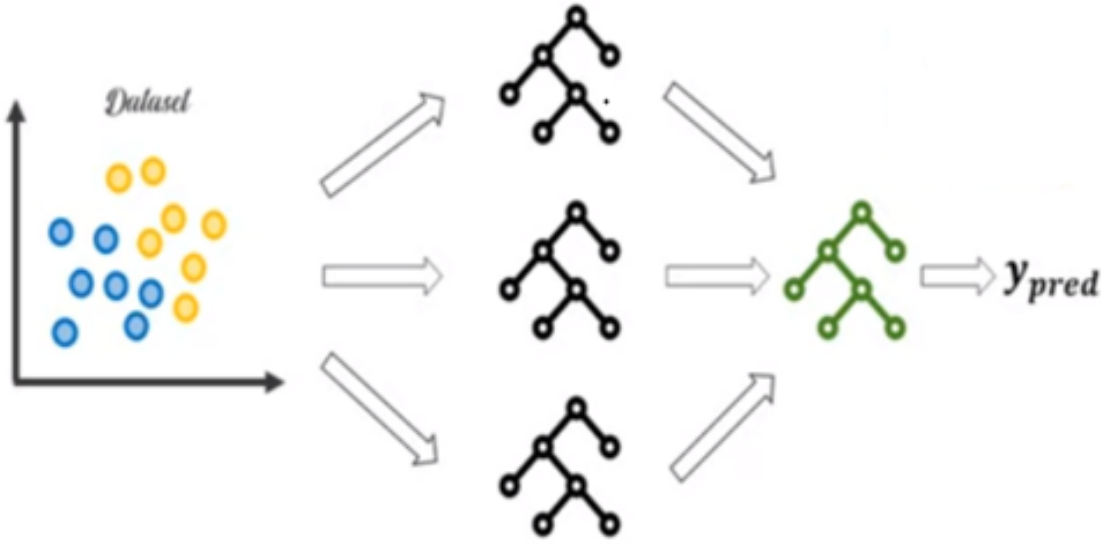
   Les modèles de base dans le Stacking peuvent être de types différents (réseaux de neurones, arbres de décision, régressions linéaires, etc.).



Bagging.



Boosting.



Stacking.

3. Quel algorithme choisir

Le choix d'une technique adaptée à la tâche à traiter peut être l'une des parties les plus difficiles. Pour n'importe quelle tâche, il existe des dizaines de techniques qui pourraient représenter un bon choix, et trouver celle qui est optimale pourrait ne pas être évident. En général, les praticiens ne devraient pas passer trop de temps sur le choix de l'algorithme optimal. Le développement de solutions basées sur l'IA est un processus itératif. La connaissance et l'expérience de différents algorithmes peuvent vous aider à développer une meilleure intuition afin de réduire le processus d'itération, mais il est rare de sélectionner immédiatement l'algorithme optimal, les paramètres et la configuration de l'apprentissage. En général, voici quelques questions que vous devriez vous poser lorsque vous êtes confronté à la sélection d'un algorithme de l'IA :

- Quelle est la taille de votre jeu d'apprentissage ?
- Prédisez-vous une catégorie d'échantillon ou une valeur quantitative ?
- Avez-vous des données étiquetées ? De quelle quantité de données étiquetées disposez-vous ?
- Connaissez-vous le nombre de catégories des résultats ?
- De combien de temps et de ressources disposez-vous pour entraîner le modèle ?
- De combien de temps et de ressources disposez-vous pour faire des prédictions ?

4. Principe de fonctionnement des NIDS basés sur l'IA

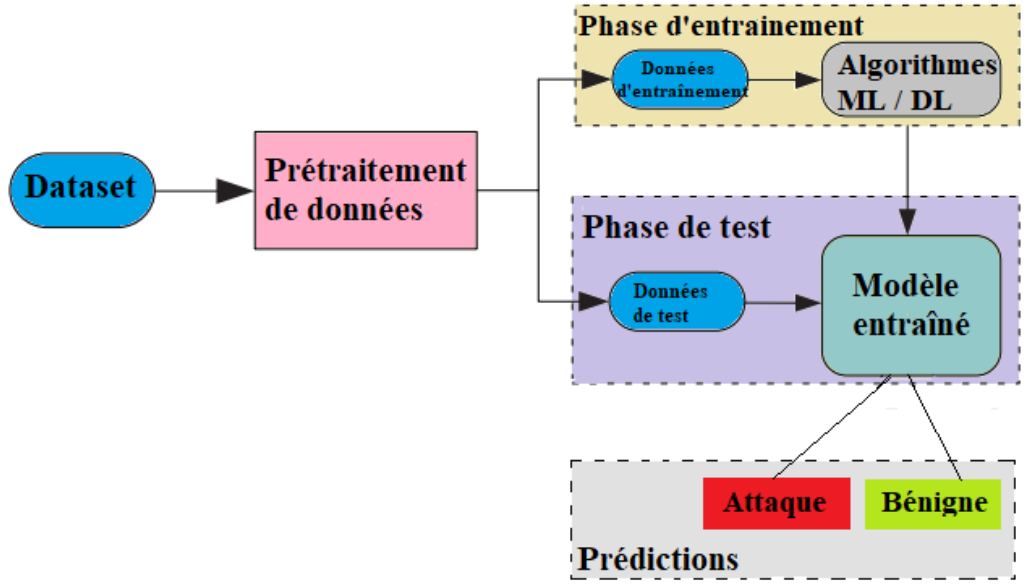
Un NIDS développé à l'aide de méthodes de ML et DL se déroule généralement en trois grandes étapes, comme le montre la figure *(cf. p.13)*, à savoir (i) la phase de prétraitement des données, (ii) la phase d'apprentissage et (iii) la phase de test.

L'ensemble de données est d'abord pré-traité afin de le transformer dans un format approprié pour être utilisé par l'algorithme sélectionné. Les données pré-traitées sont ensuite divisées de manière aléatoire en deux parties, l'ensemble de données d'entraînement et l'ensemble de données de test. En général, l'ensemble de données d'apprentissage représente près de 80% de la taille de l'ensemble de données original et les 20% restants constituent l'ensemble de données de test. L'algorithme de ML ou DL est ensuite entraîné en utilisant le dataset d'entraînement dans la phase d'entraînement. Le temps d'apprentissage de l'algorithme dépend de la taille de l'ensemble de données et de la complexité du modèle choisi. Normalement, le temps d'apprentissage des modèles de DL est plus long en raison de leur structure profonde et complexe. Une fois le modèle entraîné, il est testé à l'aide de l'ensemble de données de test et évalué en fonction des prédictions qu'il a faites. Dans le cas des modèles NIDS, l'instance de trafic réseau sera prédite comme appartenant à la classe bénigne (normale) ou à la classe d'attaque.

- **Prétraitement de données :** En Machine Learning, nous développons un modèle à partir de données, que nous sauvegardons dans un tableau appelé Dataset. Dans l'apprentissage supervisé, le Dataset contient les questions (x) et les réponses (y) au problème que la machine doit résoudre, et dans l'apprentissage non supervisé, le Dataset contient les questions (x) sans les réponses (y) au problème que la machine doit résoudre.

  Pour effectuer une analyse des données avec précision et efficacité, le prétraitement des données joue un rôle majeur. Différentes techniques sont utilisées pour pré-traiter les données, à savoir, échantillonnage, discrétisation de données, sélection des attributs, etc. Lors de l'étape d'échantillonnage, un sous-ensemble d'échantillons est choisi parmi une grande population. La discrétisation de données est utilisée pour convertir des variables continues en différents intervalles distincts. La sélection des attributs permet de garder les attributs qui sont utiles : soient ceux-ci sont pertinents, et il est important de les garder, soient ils sont tellement liés à la classe qu'à eux seuls ils emportent la décision.

- **Entraînement :** Dans cette étape, l'ensemble de données d'apprentissage est utilisé pour former, ou ajuster le modèle conçu. Un développeur alimente son modèle avec un ensemble de données d'apprentissage afin qu'il puisse "apprendre" tout ce dont il a besoin sur le type de données qu'il va analyser.
- **Validation :** Dans cette étape, l'ensemble de validation est utilisé pour une évaluation du modèle lors du réglage des hyper-paramètres. Par exemple, lorsque vous souhaitez trouver le nombre optimal de neurones dans un réseau neuronal ou le meilleur noyau pour une SVM, vous expérimentez différentes valeurs. Pour chaque réglage envisagé des hyper-paramètres, vous ajustez le modèle avec l'ensemble d'apprentissage et évaluez ses performances avec l'ensemble de validation.
- **Détection :** Dans cette étape, l'ensemble de données de test est nécessaire pour une évaluation du modèle final. Vous ne devez pas l'utiliser pour l'ajustement ou la validation.



Méthodologie généralisée des NIDS basés sur ML/DL.

Détection d'intrusion

Ici, nous montrons comment construire un classifieur d'attaques réseau en utilisant l'apprentissage automatique. Le jeu de données que nous utiliserons est le jeu appelé NSL-KDD, qui constitue une amélioration par rapport au jeu de données classique de détection des intrusions dans les réseaux largement utilisé par les professionnels de la science des données spécialisés en sécurité. Les éléments ont été collectés sur une période de 9 semaines et consistent en des données de trafic dans un réseau local (LAN). Certaines attaques de réseau ont été délibérément menées pendant la période d'enregistrement. Il y avait 38 types différents d'attaques, mais 24 seulement sont disponibles dans le jeu d'entraînement. Ces attaques appartiennent à quatre catégories générales : DoS (Déni de service), r2l (Accès non autorisés à partir de serveurs distants), u2r (Tentatives d'élévation des privilèges), probe (Attaques par force brute).

1. NIDS basés sur ML/DL

Tous les algorithmes de ML sont implémentés avec une architecture orientée objet dans laquelle chaque modèle dispose de sa propre classe. Pour créer un modèle, on génère on objet de la classe correspondante à ce modèle (c'est ce qu'on appel un estimateur). Les quatre instructions suivantes nous permettent de concevoir un modèle basé sur le ML :

1. Selectionner un estimateur et préciser ces hyperparamètres :

   Model = estimateur(hyperparametres)

   Une fois qu'on a initialisé notre modèle, on va pouvoir l'entraîner, l'évaluer, et l'utiliser en utilisant trois méthodes qu'on trouve dans toutes les classe de Scikitlearn.

2. Entraîner le modèle sur les données x, y :

   *Model.**fit(x,y)***

   Une fois notre modèle est entrainé, on peut l'évaluer :

3. Évaluer le modèle :

   *Model.**score(x,y)***

   Dans cette méthode, on fait une nouvelle fois passer les données x et y, et cette fois-ci la machine utilise les données x pour faire des prédictions et comparer et les comparées aux valeurs y.

   Une fois on est satisfait par les performance de notre modèle, alors on peut l'utilisé pour faire une nouvelle prédiction en utilisant la méthode predict () :

4. Utiliser le modèle :

   *Model.**predict(x)***

Remarque

C'est les quatre (04) lignes de code que vous devez utiliser pour développer des modèles ML.

2. Préparation des données

Avant de pouvoir appliquer des algorithmes aux données, nous devons préparer celles-ci. Nous divisons le dataset en données d'entraînement et de test comme suit :

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)

Si vous avez divisez vos données avec un autre outils, il suffit de les importer :

import pandas as pd

train_dataset = pd.read_excel("chemin vers le dataset")

test_dataset = pd.read_excel("chemin vers le dataset")

où, *train_dataset* et *test_dataset* sont, respectivement, les données d'apprentissage et les données de test.

3. Classification

Étant donné que nous avons accès à environ 126000 échantillons de données étiquetées, les méthodes d'apprentissage supervisé semblent constituer un bon point de départ. Notre tâche consiste à concevoir un classifieur qui catégorise chaque échantillon individuel dans l'une des cinq classes suivantes : bénigne, dos, r2l, u2r ou probe. Dans les scénarios pratiques, l'exactitude du modèle n'est pas le seul facteur dont vous devez tenir compte. Avec un jeu d'apprentissage de grande taille, l'efficacité de la phase d'apprentissage et la durée d'exécution sont également des facteurs importants. Les arbres de décision sont de bons points de départ, car ils sont invariants quant au rééchelonnement des données (prétraitement), et sont relativement robustes face à des caractéristiques non informatives, ce qui donne généralement de bonnes performances d'apprentissage.

3.1. NIDS basé sur les arbres de décision

Commençons par injecter nos données dans un arbre de décision, sklearn.tree.DecisionTreeClassifier :

- Selectionner un estimateur de la classe correspondante à au modèle arbres de décision :

  from sklearn.tree import DecisionT reeClassifier,

  classifier = DecisionT reeClassifier()

- Entraîner le modèle :

  classifier.fit(x_train,y_train)

- Utiliser notre modèle :

  y_pred = classifier.predict(x_test)

Pour avoir une idée approximative de la façon dont l'algorithme fonctionne, nous examinerons la matrice de confusion, en utilisant *sklearn.metrics.confusion_matrix*, et le taux d'erreur, en utilisant *sklearn.metrics.zero_one_loss*.

from sklearn.metrics import confusion_matrix

results = confusion_matrix(y_test,y_pred)

from sklearn.metrics import zero_one_loss

error = zero_one_loss(y_test,y_pred)

La matrice est comme suit :

```
> # Confusion matrix:
[[9357   59  291    3    1]
 [1467 6071   98    0    0]
 [ 696  214 1511    0    0]
 [2325    4   16  219   12]
 [ 176    0    2    7   15]]
```

L'erreur de classification est :

```
> # Error:
0.238245209368
```

Avec quelques lignes de code (et aucun réglage), obtenir une exactitude de 76,2 % (1 - taux d'erreur) dans un problème de classification à cinq classes. Ce n'est pas trop mauvais ! Cependant, ce nombre n'a pas beaucoup de sens. Pour voir si ce problème peut être causé par le choix de l'arbre de décision comme classifieur, regardons ce qui se passe avec l'algorithme k-NN.

3.2. NIDS basé sur k-NN

- Selectionner un estimateur de la classe correspondante au modèle k-NN :

  from sklearn.neighbors import KNeighborsClassifier

  classifier = KNeighborsClassifier(n_neighbors = 1)

- Entraîner le modèle :

  classifier.fit(x_train,y_train)

- Utiliser notre modèle :

  y_pred = classifier.predict(x_test)

Pour avoir une idée approximative de la façon dont l'algorithme fonctionne, nous examinerons la matrice de confusion, en utilisant :

from sklearn.metrics import confusion_matrix

results = confusion_matrix(y_test,y_pred)

from sklearn.metrics import zero_one_loss

error = zero_one_loss(y_test,y_pred)

La matrice est comme suit :

```
> # Confusion matrix:
[[9455   57  193    2    4]
 [1675 5894   67    0    0]
 [ 668  156 1597    0    0]
 [2346    2   37  151   40]
 [ 177    0    4    6   13]]
```

L'erreur de classification est :

```
> # Error:
0.240951029099
```

Le taux d'erreur est à peu près le même, et la matrice de confusion qui en résultent présente des caractéristiques très semblables à celles que nous avons vues pour l'arbre de décision. Pour voir si ce problème peut être causé par le choix du k-NN comme classifieur, regardons ce qui se passe avec l'algorithme SVM.

3.3. NIDS basé sur SVM

- Selectionner un estimateur de la classe correspondante au modèle SVM :

  from sklearn.svm import LinearSVC

  classifier = LinearSVC()

- Entraîner le modèle :

  classifier.fit(x_train,y_train)

- Utiliser notre modèle :

  y_pred = classifier.predict(x_test)

Pour avoir une idée approximative de la façon dont l'algorithme fonctionne, nous examinerons la matrice de confusion, en utilisant :

from sklearn.metrics import confusion_matrix

results = confusion_matrix(y_test,y_pred)

from sklearn.metrics import zero_one_loss

error = zero_one_loss(y_test,y_pred)

La matrice est comme suit :

```
> # Confusion matrix:
[[9006  294  405    3    3]
 [1966 5660   10    0    0]
 [ 714  122 1497   88    0]
 [2464    2    1  109    0]
 [ 175    1    0    8   16]]
```

L'erreur de classification est :

```
> # Error:
0.278167139815
```

Le taux d'erreur est à peu près le même, et la matrice de confusion qui en résultent présente des caractéristiques très semblables à celles que nous avons vues pour l'arbre de décision et k-NN. Pour voir si ce problème peut être causé par le choix des algorithmes supervisés, regardons ce qui se passe avec les algorithmes non supervisés.

3.4. NIDS basé sur k-means

L'application du clustering sur le jeu de données NSL-KDD nous permettra de voir rapidement en quoi cette technique est différente des méthodes supervisées que nous avons appliquées auparavant. Dans notre exemple, nous savons que nos données contiennent cinq catégories d'échantillons, donc nous choisirons l'algorithme de clustering k-means pour la tâche, avec k (le nombre de clusters) égal à 5.

- Selectionner un estimateur de la classe correspondante au modèle k-means :

  from sklearn.cluster import KMeans

classifier = KMeans(n_clusters = 5,max_iter = 300)

- Entraîner le modèle :

  classifier.fit(x_train)

- Utiliser notre modèle :

  y_pred = classifier.predict(x_test)

- Inspecte les résultats du clustering :

  print(pd.Series(y_pred).value_counts())

```
>
1   15262
2    5211
0    2069
3       2
```

3.5. NIDS basé sur méthode ensemble

Dans notre exemple, nous utilisons l'algorithme de Vote majoritaire. Les techniques à entraîner sont un classifieur d'arbre de décision, SVM et kNN.

- Selectionner les estimateur de la classe correspondante au modèles selectionnés :

  from sklearn.ensemble import V otingClassifier

  from sklearn.tree import DecisionT reeClassifier

  from sklearn.svm import SV C

  from sklearn.neighbors import KNeighborsClassifier

  dt = DecisionT reeClassifier()

  svm = LinearSV C()

  knn = KNeighborsClassifier(n_neighbors = 1)

  voc = V otingClassifier(estimators = [('dt', dt), ('svm',svm)('knn',knn)],voting =' hard')

- Entraîner le modèle :

  voc.fit(x_train,y_train)

- Utiliser notre modèle :

  y_pred = voc.predict(x_test)

Pour avoir une idée approximative de la façon dont l'algorithme fonctionne, nous examinerons la matrice de confusion, en utilisant :

from sklearn.metrics import confusion_matrix

results = confusion_matrix(y_test,y_pred)

from sklearn.metrics import zero_one_loss

error = zero_one_loss(y_test,y_pred)

La matrice est comme suit :

```
[[9448   42  217    2    2]
 [2098 5475   63    0    0]
 [ 531  181 1711    0    0]
 [2409    0  124   39    2]
 [ 146    0   51    1    2]]
```

L'erreur de classification est :

```
0.2603353442157559
```

Qu'est ce qui est important pour les systèmes ML dans le domaine de la sécurité

Les éléments suivant sont particulièrement les éléments les plus importants pour les systèmes de l'apprentissage automatique dans le domaine de la sécurité : **Qualité des données** et **Qualité du modèle**.

- **Qualité des données :** la qualité des données passée en entrée dans un système d'apprentissage automatique détermine son succès ou son échec.
    1. **Problème : biais dans les jeux de données :** Les dataset bien équilibrés sont rares, et l'utilisation de datasets non équilibrés peut se traduire par des biais difficiles à détecter. Par exemple, Lors de l'entraînement d'un classifieur de spams de manière supervisée, injecter dans un algorithme des données d'apprentissage composées de spams qui ne contiennent que des données de santé et des publicités médicales ne permettra pas d'obtenir un modèle équilibré et généralisable. Par conséquent, le système qui en résultera sera peut-être efficace pour reconnaître les courriels non sollicités faisant la promotion de médicaments, mais il sera probablement incapable de détecter les courriels de contenu explicites pour adultes.
    2. **Problème : inexactitude des étiquettes :** Lors d'un apprentissage supervisé, des données mal étiquetées risquent sérieusement d'entraîner une perte d'exactitude des algorithmes. L'exactitude de la validation lors de la phase de développement peut sembler prometteuse, mais le modèle ne fonctionnera probablement pas comme prévu lorsqu'il sera alimenté avec des données réelles. Vérifier l'exactitude des étiquettes dans un jeu de données nécessite souvent de coûteuses ressources en termes d'expertise humaine.
    3. **Solutions : qualité des données :** L'étape la plus critique dans le traitement des problèmes de qualité des données pour les systèmes de ML dans le domaine de la sécurité est de reconnaître l'existence du problème.
        - *Le déséquilibre des classes :* Est une manifestation d'un biais dans les données. Ce déséquilibre est un problème assez évident que l'on peut détecter au cours de la phase d'apprentissage, et que l'on peut atténuer par un **sur-échantillonnage** et un **sous-échantillonnage**. Dans la cas de l'apprentissage non supervisé, on opte pour la réduction de la dimensionnalité.
        - Dans certains cas, vous pouvez éviter les problèmes de qualité des données en définissant soigneusement **l'ampleur du problème**. Par exemple, les systèmes qui prétendent détecter toutes sortes d'e-mails d'hameçonnage auront du mal à générer un dataset d'apprentissage représentatif. Cependant, si l'ampleur de la tâche se limite au problème le plus important auquel une organisation est confrontée – par exemple, détecter les e-mails d'hameçonnage qui tentent de dérober les clics de l'utilisateur– il sera plus facile de recueillir des données plus ciblées sur ce problème.
        - Des étiquettes inexactes produites par des erreurs humaines peuvent être rendues moins probables en impliquant **plusieurs annotateurs indépendants** dans le processus d'étiquetage. En supposant que les annotateurs humains n'ont pas attribué les étiquettes de façon insuffisamment sérieuse, le niveau de désaccord

entre eux pour l'étiquetage d'un échantillon particulier est aussi souvent utilisé comme étant la limite supérieure de la probabilité pour qu'un classifieur soit capable de prédire l'étiquette correcte de cet échantillon. Donc, Il est préférable d'exclure ceux-ci du jeu de données pour éviter de dérouter les objectifs d'apprentissage de l'algorithme.

- **Qualité du modèle :**
    1. **Problème : optimisation des hyper-paramètres :** Les débutants essaient généralement d'éviter la complexité en se contentant des valeurs par défaut fournies par la bibliothèque d'apprentissage automatique. De nombreuses bibliothèques d'apprentissage automatique matures (y compris scikit-learn) fournissent des valeurs par défaut qui sont soigneusement choisies et qui conviennent pour la majorité des cas d'utilisation. Cependant, il n'est évidemment pas possible qu'un ensemble d'hyper-paramètres soit optimal pour tous les scénarios.
    2. **Solutions : optimisation des hyper-paramètres :**
        - **Bien comprendre l'algorithme et ses paramètres :** Bien comprendre l'algorithme et faire des expériences quant à son implémentation peut vous guider dans le processus itératif d'optimisation manuelle des hyper-paramètres. Cependant, même si vous êtes nouveaux venus sur ce terrain, le processus de réglage n'a pas besoin d'être complètement fait à l'aveugle. La visualisation des résultats de l'apprentissage peut généralement entraîner des ajustements des hyperparamètres dans certaines directions.
        - **Imiter des modèles similaires :** Une autre façon courante de résoudre ce problème consiste à faire des recherches sur des travaux antérieurs qui soient similaires dans le domaine concerné. Même si le problème n'est pas exactement de la même nature, copier des hyper-paramètres d'autres travaux, tout en comprenant pourquoi ces choix ont été faits, peut vous faire gagner beaucoup de temps car quelqu'un d'autre a déjà travaillé à la résolution d'un problème similaire.
        - **Ne commencez pas à régler les paramètres trop tôt :** Ne vous préoccupez des paramètres que lorsque vous soupçonnez qu'ils pourraient être la cause d'un problème dans votre classifieur. Commencer par les configurations les plus simples et surveiller les améliorations potentielles à apporter en cours de route est généralement une bonne pratique à suivre.

Complément

Il n'y a pas de règles strictes sur la façon d'obtenir de meilleures performances en choisissant certains algorithmes plutôt que d'autres. Cependant, voici une liste d'astuces qui pourraient vous être utiles dans votre processus de prise de décision :

- Avoir moins de caractéristiques signifie avoir à effectuer moins d'opérations arithmétiques, ce qui peut améliorer les performances. L'application de méthodes de réduction de la dimensionnalité pour supprimer les caractéristiques inutiles de votre jeu de données peut également améliorer les performances ;
- Les modèles arborescents (par exemple, les arbres de décision ou les forêts aléatoires) ont tendance à avoir une très bonne performance de prédiction, parce ils sont invariants quant au prétraitement des données, et sont relativement robustes face à des caractéristiques non informatives ;
- Les SVM sont l'une des familles de modèles les plus lentes à entraîner, et sont également très gourmands en mémoire ;

- Les algorithmes de DL sont lents à entraîner, et nécessitent beaucoup de ressources (au moins des millions de multiplications matricielles sont généralement mises en œuvre). En revanche, ils peuvent facilement être parallélisés avec un matériel approprié, par exemple des processeurs graphiques (GPU).

Exercices

1. Exercice

[solution n°1 p. 24]

Un NIDS peut être basé sur l'apprentissage automatique (Machine Learning) ou ne pas l'être. Quelle est la principale différence entre les deux ?

2. Exercice

[solution n°2 p. 24]

Quelles sont les étapes principales des NIDS basés sur ML/DL ?

3. Exercice

[solution n°3 p. 24]

C'est quoi le rôle de la phase d'entraînement ?

4. Exercice

[solution n°4 p. 24]

Les instructions permettant de concevoir un modèle basé sur le ML sont :

- ☐ Sélectionner un estimateur
- ☐ Entraîner le modèle
- ☐ Évaluer le modèle
- ☐ Comparer la performance du modèle à un autre modèle
- ☐ Utiliser le modèle

5. Exercice

[solution n°5 p. 25]

Qu'est-ce qui permet de résoudre le problème de biais dans les données à passer en entrée dans un système d'apprentissage automatique ?

- ☐ Échantillonnage et Sur-échantillonnage
- ☐ Pré-traitement
- ☐ Définition de l'ampleur du problème

Solutions des exercices

Solution n°1

[exercice p. 23]

Un NIDS peut être basé sur l'apprentissage automatique (Machine Learning) ou ne pas l'être. Quelle est la principale différence entre les deux ?

La principale différence entre les deux réside dans leur approche de détection des intrusions et leur capacité à s'adapter à de nouvelles attaques inconnues.

Solution n°2

[exercice p. 23]

Quelles sont les étapes principales des NIDS basés sur ML/DL ?

Le prétraitement des données, la phase d'apprentissage, la phase de validation et la phase de test

Solution n°3

[exercice p. 23]

C'est quoi le rôle de la phase d'entraînement ?

Dans la phase de formation (apprentissage), un développeur alimente son modèle avec un ensemble de données sélectionnées afin que celui-ci puisse "apprendre" tout ce dont il a besoin sur le type de données qu'il va analyser.

Solution n°4

[exercice p. 23]

Les instructions permettant de concevoir un modèle basé sur le ML sont :

- ☑ Sélectionner un estimateur
- ☑ Entraîner le modèle
- ☑ Évaluer le modèle
- ☐ Comparer la performance du modèle à un autre modèle
- ☑ Utiliser le modèle

Solution n°5

[exercice p. 23]

Qu'est-ce qui permet de résoudre le problème de biais dans les données à passer en entrée dans un système d'apprentissage automatique ?

- ☑ Échantillonnage et Sur-échantillonnage
- ☐ Pré-traitement
- ☑ Définition de l'ampleur du problème

Bibliographie

Stamp, M. (2017). Introduction to machine learning with applications in information security. Chapman and Hall/CRC.

Halder, S., & Ozdemir, S. (2018). Hands-On Machine Learning for Cybersecurity : Safeguard your system by making your machines intelligent using the Python ecosystem. Packt Publishing Ltd.

Parisi, A. (2019). Hands-on artificial intelligence for cybersecurity : Implement smart AI systems for preventing cyber attacks and detecting threats and network anomalies. Packt Publishing Ltd.

Sengupta, N., & Sil, J. (2020). Intrusion Detection : A Data Mining Approach. Springer Nature.

Charniak, E. (2021). Introduction au deep learning. Dunod.

Bouaziz, M. (2017). Réseaux de neurones récurrents pour la classification de séquences dans des flux audiovisuels parallèles (Doctoral dissertation, Université d'Avignon).

Goodfellow, I. (2016). Deep Learning-Ian Goodfellow, Yoshua Bengio, Aaron Courville- Google Books.

Mandic, D., & Chambers, J. (2001). Recurrent neural networks for prediction : learning algorithms, architectures and stability. Wiley.

Touzet, C. (1992). Les réseaux de neurones artificiels : Introduction au connexionnisme : Cours. Exercices et travaux pratiques. (Disponible online :) http ://www.touzet.org/Claude/Web- Fac-Claude/Les_reseaux_de_neurones_artificiels.pdf