

RAPPORT PROJET UPEC - ESME



Détection d'intrusion sur un réseau à partir
d'algorithmes de Machine Learning et Deep Learning

Lien GitHub : <https://github.com/CecileKafrouni/IntrusionDetection.git>

Année scolaire : 2020 – 2021

Responsable de projet : Yue Li

Table des matières

Introduction *	5
I. Motivation et objectifs *	7
II. Etat de l'art *	9
III. Active Learning *	12
A. Qu'est-ce qu'un apprentissage actif ? *	13
B. Scénarios *	15
C. Exemple d'étapes d'utilisation d'apprentissage actif *	17
IV. Préparation des données *	20
A. Analyser la data *	21
B. Ajouter une colonne cible *	22
C. Supprimer les colonnes vides ou inutiles *	22
D. Équilibrage des données *	23
E. Nettoyage des données *	24
F. Conversion de l'adresse IP *	24
G. Normalisation des données *	25

V. Machine Learning	26
A. Théorie *	27
B. Métriques *	30
C. K-Nearest Neighbors *	34
D. Gaussian Naives Bayes **	35
E. Support Vector Machine *	40
F. Perceptron **	41
G. Decision Tree Classifier *	48
H. Random Forest Classifier *	53
I. XGBoost Classifier *	57
VI. Deep Learning *	60
A. Théorie *	61
B. Choix des modèles *	61
C. Les hyperparamètres *	63
D. CNN 1D *	64
E. CNN 2D *	65
F. LSTM *	67
G. Résultats et comparaisons *	68
VII. Interfaces *	71
A. Application en local PySimpleGUI *	72
B. Application Web Flask *	79
Conclusion *	89

*Chanty *Walid *Théodore *Cécile

Remerciements

Nous tenons à remercier M. Li, pour nous avoir suivi, accompagné et aidé chaque semaine depuis le début du projet.

Merci également à M. Dandoush pour sa supervision et de nous avoir fourni du matériel pour mener à bien ce projet.

Merci à tous ceux qui ont contribué à ce projet.

Introduction

Les cyber-attaques sont considérées comme une nouvelle arme à distance ciblant les infrastructures critiques telles qu'une campagne présidentielle, un programme nucléaire, les données du personnel gouvernemental et les fournisseurs de logiciels.

Il est essentiel de distinguer les trafics nuisibles des trafics normaux tout en utilisant efficacement le réseau Internet. La sécurisation du système DNS contre tout accès non autorisé est d'une importance capitale pour le fonctionnement des réseaux privés et de l'internet. Comme les pirates utilisent des méthodologies sophistiquées pour attaquer les demandes et les réponses du DNS, le DNS sur le protocole HTTPS est introduit en cryptant les requêtes du DNS et en les transmettant dans un canal caché. Cette approche améliore la protection de la vie privée et permet de surmonter certaines des vulnérabilités du DNS, telles que les attaques de type "man-in-the-middle". Un système de détection d'intrusion (IDS) joue un rôle important dans la surveillance du trafic des appareils connectés à Internet et dans la détection des attaques pour le trafic du ministère de la santé dans une topologie de réseau.

L'IDS est l'outil de défense le plus critique contre les attaques de réseau sophistiquées et toujours plus nombreuses.

I. Motivation et objectifs

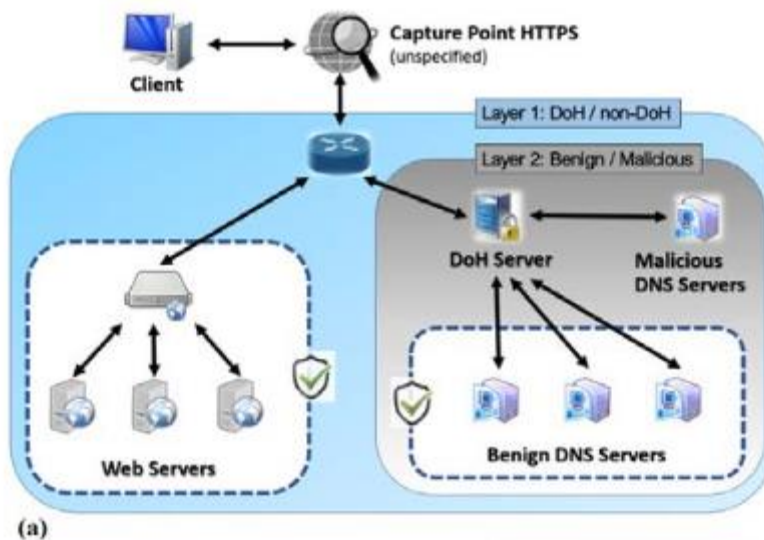
La tâche du projet consiste à déterminer, à partir d'un ensemble d'informations statistiques d'un flux réseau, si ce flux est un trafic bénin ou une intrusion, sur la base de différents modèles d'apprentissages supervisés formé à partir de ses informations statiques.

On utilisera le machine Learning afin de détecter si le protocole est bien un Doh ou non. Tandis que avec le Deep Learning, nous déterminerons si le trafic est bénin ou malicieux.

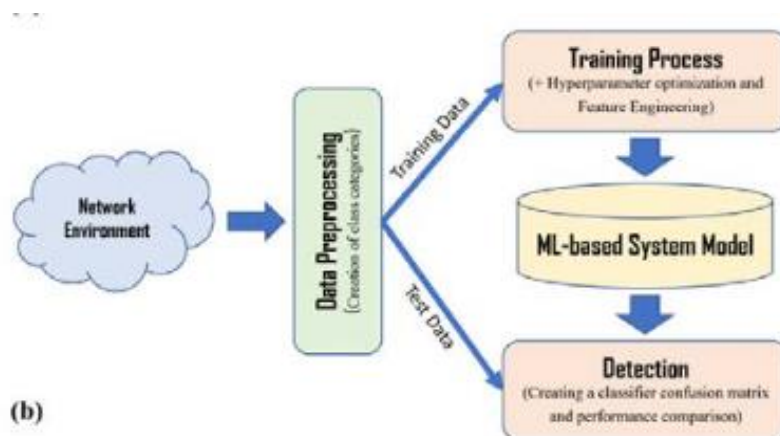
II. Etat de l'art

Une approche systématique est proposée pour évaluer la capacité de différents algorithmes d'apprentissage machine à être utilisés pour analyser, tester et évaluer le trafic du ministère de la santé dans les canaux et tunnels clandestins.

La collecte des données du réseau et processus de formation se comprend en analysant ces figures ci-dessous :



La figure représente la procédure de formation à la détection d'intrusion comprenant le prétraitement des données, la formation et l'optimisation des algorithmes de formation, le déploiement de classificateurs basés sur le machine learning et le test du modèle pour extraire les mesures de performance de la classification.



Cette figure représente la topologie du réseau utilisée pour capturer les ensembles de données de trafic, y compris le trafic bénin et malveillant, ainsi que le trafic autre que celui donné.

Une recherche qui se concentre sur les classificateurs de séries temporelles pour détecter et caractériser le trafic de drogue dans une approche d'apprentissage automatique à deux niveaux qui déploie la drogue dans une application et distingue le trafic bénin et malveillant de la drogue.

Récemment, l'Institut canadien de cybersécurité (CIC) a publié un ensemble de données qui comprend l'implémentation du protocole DoH dans une application utilisant cinq navigateurs et outils différents et quatre serveurs pour capturer le trafic bénin, malveillant et non-bienveillant du DoH. Dans l'approche à deux niveaux utilisés pour capturer le trafic DoH bénin et malveillant ainsi que le trafic non-DoH, le premier niveau est utilisé pour classer le trafic DoH du trafic non-DoH, et le deuxième niveau est utilisé pour caractériser le trafic DoH bénin du trafic DoH malveillant.

Parmi les algorithmes de classification supervisés nous avons utilisé :

- Decision Tree Classifier (DTC) ,
- Nearest Neighbors (KNN) ,
- Gaussian Naive Bayes (GNB) ,
- XGBoost (XGB) ,
- Random Forest Classifier (RFC).
- Support vector Machines (SVM)

Nous allons donc faire intervenir le machine learning afin de pouvoir gérer tout ceci.

III. Active Learning

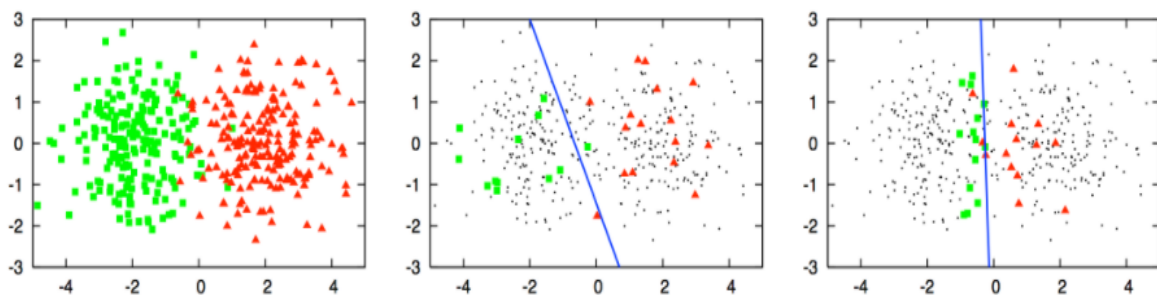
A. Qu'est-ce qu'un apprentissage actif ?

L'apprentissage actif est un modèle d'apprentissage semi-supervisé où un oracle intervient au cours du processus. Plus précisément, contrairement au cadre classique où les données sont connues et imposées, en apprentissage actif, c'est l'algorithme d'apprentissage qui demande des informations pour des données précises. L'hypothèse principale dans l'apprentissage actif est que si un algorithme d'apprentissage peut choisir les données qu'il veut apprendre, il peut être plus performant que les méthodes traditionnelles avec beaucoup moins de données pour la formation.

Cette technique repose sur l'hypothèse que l'acquisition de données non étiquetées est beaucoup moins coûteuse que celle de données étiquetées. Elle repose également sur l'hypothèse que l'apprentissage est plus efficace lorsque l'on est *curieux*, en cherchant les données les plus intéressantes à étiqueter. L'apprentissage actif permet d'effectuer diverses tâches classiques dans le domaine de l'apprentissage automatique telles que la classification ou la régression. Contrairement à l'apprentissage passif, c'est l'apprenant (*learner*) qui choisit les données à étiqueter par l'oracle.

L'apprentissage actif permet d'obtenir de bons résultats avec un ensemble de données étiquetées réduit.

Voici un exemple concret pour comprendre l'apprentissage actif



En regardant l'image la plus à gauche ci-dessus (tirée de cette enquête), nous avons deux grappes, celles de couleur verte et celles de couleur rouge. Cependant, nous pouvons supposer que nous ne connaissons pas les étiquettes (rouge ou vert) des points de données, mais essayer de trouver l'étiquette pour chacun d'eux serait très coûteux. Par conséquent, nous souhaitons échantillonner un petit sous-ensemble de points et trouver ces étiquettes et utiliser ces points de données étiquetés comme données de formation pour un classificateur.

Dans l'image du milieu, la régression logistique est utilisée pour classer les formes en échantillonnant d'abord au hasard un petit sous-ensemble de points et en les étiquetant. Toutefois, nous voyons que la limite de décision créée à l'aide de la régression logistique (la ligne bleue) est sous-optimale. Cette ligne est clairement faussée loin des points de données rouges et dans la zone des formes vertes. Cela signifie qu'il y aura de nombreux points de données verts qui seront étiquetés incorrectement comme rouges.

Dans l'image la plus juste, la régression logistique est utilisée à nouveau, mais cette fois, nous avons sélectionné un petit sous-ensemble de points à l'aide d'une méthode de requête d'apprentissage active. Cette nouvelle limite de décision est nettement meilleure car elle sépare mieux les deux couleurs. Cette amélioration provient de la sélection de points de données supérieurs de sorte que le classificateur a été en mesure de créer une très bonne limite de décision.

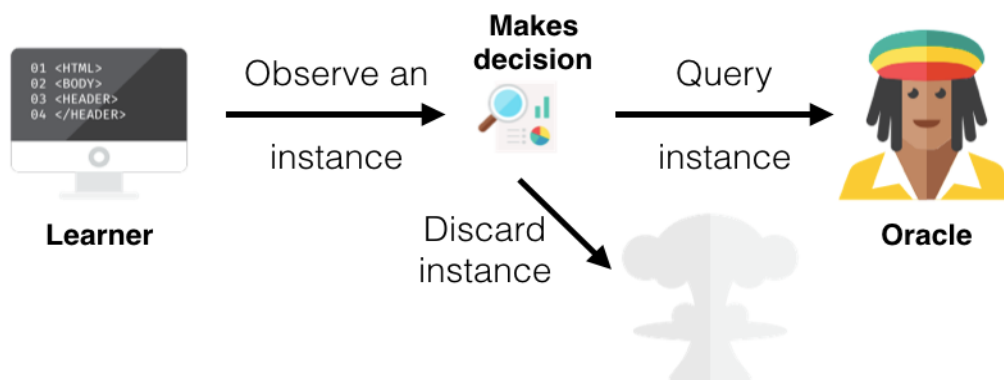
B. Scénarios

Dans l'apprentissage actif, il existe généralement trois scénarios ou paramètres dans lesquels l'apprenant interrogera les étiquettes des instances. Les trois principaux scénarios qui ont été pris en compte dans la littérature sont les suivants :

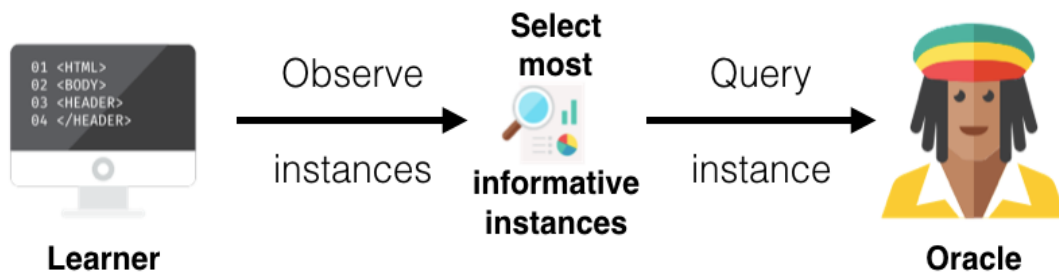
Synthèse des requêtes d'adhésion : il s'agit d'un grand terme qui signifie simplement que l'apprenant génère/construit une instance (à partir d'une certaine distribution naturelle sous-jacente). Par exemple, si les données sont des images de chiffres, l'apprenant créerait une image similaire à un chiffre (elle peut être tournée ou une partie du chiffre exclue) et cette image créée est envoyée à l'oracle pour l'étiqueter.



Échantillonnage sélectif basé sur le flux : dans ce paramètre, vous faites l'hypothèse que l'obtention d'une instance non étiquetée est gratuite. Sur la base de cette hypothèse, vous sélectionnez ensuite chaque instance non étiquetée une à la fois et permettez à l'apprenant de déterminer s'il veut interroger l'étiquette de l'instance ou la rejeter en fonction de son informative. Pour déterminer l'information de l'instance, vous utilisez une stratégie de requête. En suivant l'exemple ci-dessus, vous sélectionnez une image de l'ensemble d'images non étiquetées, déterminez si elle doit être étiquetée ou jetée, puis répétez avec l'image suivante.



Échantillonnage en commun : ce paramètre suppose qu'il existe un grand **bassin** de données non étiquetées, comme c'est le cas pour l'échantillonnage sélectif en cours d'eau. Les instances sont ensuite tirées de la piscine selon une certaine mesure d'information. Cette mesure s'applique à toutes les instances de la piscine (ou à un sous-ensemble si le pool est très grand) puis les instances les plus informatives sont sélectionnées. C'est le scénario le plus courant dans la communauté de l'apprentissage actif. En continuant avec l'exemple dans les deux scénarios ci-dessus, toutes les images non étiquetées des chiffres seront classées, puis la meilleure (la plus informative) instance (s) sera sélectionnée et leurs étiquettes demandées.



C. Exemple d'étapes d'utilisation d'apprentissage actif

1. Etape 0 : Recueillir les données

Il est important de s'assurer que l'ensemble de données que nous collectons est représentatif de la véritable distribution des données. En d'autres termes, essayer d'éviter beaucoup de données biaisées. En réalité, il est impossible d'avoir un échantillon totalement représentatif en raison de limitations telles que le droit, le temps ou la disponibilité.

Dans cet exemple, nous aurons les 5 points de données suivants. Les fonctionnalités A et B représentent certaines fonctionnalités qu'un point de données peut avoir. Il est important de noter que les données que nous recueillons ne sont pas étiquetées.

Cas	Caractéristique A	Caractéristique B
d ₁	10	0
d ₂	4	9
d ₃	8	5
d ₄	3	3
d ₅	5	5

2. Etape 1 : Fractionnement en ensemble de données de semence et non étiquetées

Nous devons diviser nos données en un très petit ensemble de données que nous étiquèterons et un grand ensemble de données non étiquetées. Dans la terminologie d'apprentissage actif, nous appelons ce petit ensemble de données étiquetées la graine. Il n'y a pas de nombre défini ou de pourcentage des données non étiquetées qui sont généralement utilisées. Une fois que nous avons mis de côté les données que nous utiliserons pour la graine, nous devons les étiqueter. En effet, nous sélectionnons deux instances pour la graine, d₁ et d₃. Les étiquettes possibles dans ce cas sont « Y » et « N ».

Ensemble de données semences étiquetées :

Cas	Caractéristique A	Caractéristique B	Étiquette
d ₁	10	0	Y (en)
d ₃	8	5	N

Ensemble de données non étiquetées

Cas	Caractéristique A	Caractéristique B
d ₂	4	9
d ₄	3	3
d ₅	5	5

3. Etape 2 : Former le modèle

Après avoir divisé nos données, nous utilisons la graine pour former notre apprenant comme un projet normal d'apprentissage automatique (en utilisant la validation croisée, etc.). En outre, le type d'apprenant qui est utilisé serait basé sur notre connaissance du domaine et généralement, nous utiliserions des pencheurs qui donnent une réponse probabiliste à savoir si une instance a une étiquette particulière, que nous utilisons ces probabilités pour les stratégies de requête.

Dans l'exemple, nous pouvons utiliser n'importe quel classificateur et nous nous entraînerons sur nos deux instances étiquetées.

4. Etape 3 : Choisir des instances non étiquetées

Une fois que nous avons formé notre apprenant, nous sommes maintenant prêt à sélectionner une instance ou des instances à interroger. Nous devons déterminer le type de scénario que nous souhaitons utiliser (c'est-à-dire la synthèse des requêtes d'adhésion, l'échantillonnage sélectif basé sur le flux ou l'échantillonnage basé sur le pool) et la stratégie de requête.

Nous utiliserons l'échantillonnage en piscine avec une taille de lot de 2. Cela signifie qu'à chaque itération, nous sélectionnerons deux instances de notre ensemble de données non étiquetées, puis nous ajouterons ces instances à notre jeu de données étiqueté. Nous utilisons le moins de confiance pour sélectionner nos instances. Notre maigre sélectionne d_2 et d_4 dont les étiquettes interrogées sont « Y » et « N », respectivement.

Ensemble de données étiquetées

Cas	Caractéristique A	Caractéristique B	Étiquette
d_1	10	0	Y (en)
d_3	8	5	\hat{A}_n
d_2	4	9	Y (en)
d_4	3	3	\hat{A}_n

Ensemble de données non étiquetées

Cas	Caractéristique A	Caractéristique B
d_5	5	5

5. Etape 4 : Critère d'arrêt

Dans notre exemple, nous nous arrêterons à une itération et nous en aurons donc fini avec notre algorithme d'apprentissage actif. Nous pouvons également avoir un ensemble de données de test distinct sur qui nous évaluons notre apprenant et enregistrons ses performances. De cette façon, nous pouvons voir comment nos performances sur l'ensemble de test se sont améliorées ou ont stagné avec des données étiquetées ajoutées.

IV. Préparation des données

A. Analyser la data

Une brève analyse de la donnée est nécessaire pour pouvoir réfléchir à un travail de nettoyage des données et de feature engineering. C'est le but de notre fonction « analyser_df » qui montre les informations basiques de notre dataset grâce aux fonctions de la librairie pandas :

```
def analyser_df(df):  
  
    #Analyse rapide  
    print("\nShape de La frame :\n", df.shape)  
    print("\nNoms des colonnes :\n", df.columns)  
    print("\nPremières Lignes :\n", df.head())  
    print("\nDernières Lignes :\n", df.tail())  
  
    #On regarde les différentes classes presentes dans la colonne label  
    print("\n ELEMENTS DE LA COLONNE LABEL")  
    print(df.Label.unique())  
  
    #On compte le nombre de fois que chaque element revient  
    print("\n ELEMENTS DE LA COLONNE LABEL AVEC NB DE FOIS QU'ILS REVIENNENT")  
    print(df.Label.value_counts())
```

- df.shape : dimensions de la dataset
- df.columns : nombre de colonnes de la dataset
- df.head : premières lignes de la dataset
- df.tail : dernières lignes de la dataset
- df.Label.unique() : nombre d'éléments de notre colonne cible 'Label' qui définit si une entrée de connexion est une intrusion ou non. (Nous avons donc 2 valeurs : 'intrusion' et 'begin' pour la dataset contenant les intrusions ou 'DoH' et 'nonDoH' pour la dataset contenant les types de connexion)
- df.Label.value_counts() : nombre de fois que les valeurs reviennent dans la colonne cible 'Label'

A partir de cette pré analyse des données découlent les fonctions ci-dessus pour nettoyer la dataset et donc la rendre exploitable pour nos modèles de machine learning et de deep learning

B. Ajouter une colonne cible

Pour rendre notre variable cible exploitable, il faut que les données de la dataset soient de type Int ou Float. Il faut ainsi transformer les éléments de la colonne 'Label' (qui sont de type String) en type Int ou Float

```
def rajouter_colonne(df, colonne) :  
  
    df[colonne] = 0  
    z = 0  
  
    for x in df[colonne]:  
        if (colonne == 'Intrusion'):  
            if(df.Label[z] == 'Benign'):  
                df[colonne][z] = 0  
            else :  
                df[colonne][z] = 1  
            z = z+1  
        else:  
            if(df.Label[z] == 'DoH'):  
                df[colonne][z] = 1  
  
            else :  
                df[colonne][z] = 0  
            z = z+1
```

Notre fonction « rajouter_colonne » permet donc de binariser les valeurs de la colonne cible en 0 ou 1 (puisque nous n'avons que deux valeurs possibles). On remplace donc les valeurs de la colonne cible par les valeurs binarisées de notre fonction.

C. Supprimer les colonnes vides ou inutiles

La fonction « supprimer_colonne_vide » va effectuer deux traitements :

```
def supprimer_colonne_vide(df) :  
    for colonne in df:  
        compteur = 0  
        for valeur in df[colonne]:  
            if valeur == 0:  
                compteur +=1  
  
        if (compteur == len(df)  
            or colonne == 'TimeStamp'  
            or colonne == 'index'  
            or colonne == 'Label'):  
  
            del df[colonne]
```

Supprimer les colonnes dont les valeurs de comportent que des 0

Supprimer les colonnes qui ne sont pas utiles pour l'entraînement des modèles (comme par exemple la colonne 'TimeStamp' qui renseigne sur la date ou 'index' qui représente le nombre de lignes)

D. Équilibrage des données

L'équilibrage des données est un processus très important pour le bon apprentissage d'un modèle. Une dataset non équilibrée pourrait entraîner un apprentissage moins important pour les données qui sont les moins redondantes dans une colonne.

Pour équilibrer les données, nous avons utilisé une technique d'over-sampling qui permet d'ajouter de la donnée jusqu'à ce qu'une dataset soit équilibrée : La technique SMOTE.

La technique SMOTE (synthetic minority oversampling technique) est l'une des méthodes de suréchantillonnage les plus couramment utilisées pour résoudre les problèmes de données déséquilibrées. SMOTE synthétise de nouveaux exemples de minorités entre des exemples de minorités existants. Des enregistrements d'entraînement synthétiques sont générés en sélectionnant de manière aléatoire un ou plusieurs des k plus proches voisins pour chaque exemple de la classe minoritaire.

Nous utilisons cette technique en appelant la fonction SMOTE (de la librairie sklearn) à chaque fois que nousinstancions un modèle. Comme le montre l'image ci-dessous, les données sont équilibrées dans les variables X_smote et y_smote qui représentent la variable cible et les

```
X = df.drop([colonne], axis = 1)
y = df[colonne]

oversample = SMOTE()

X_smote, y_smote = oversample.fit_sample(X, y)

X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=1)
```

features.

L'équilibrage des données ne fait pas partie du fichier contenant le data processing. Cependant, c'est une étape très importante de transformation des données pour le bon apprentissage d'un modèle.

E. Nettoyage des données

La fonction « nettoyage » consiste à enlever toutes les données qui peuvent entrainer des problèmes lors de l'entraînement d'un modèle (comme des valeurs manquantes ou encore des valeurs très grandes). Il existe plusieurs techniques permettant de traiter ces données.

```
def nettoyage(df):
    result = df.copy()
    # On remplace els donnees Nan par 0
    result.fillna(0, inplace=True)

    # On arrondie les nb avec 5 chiffres apreès la virgule
    for colonne in result.columns:
        if(df[colonne].dtypes == 'float64'):
            a = np.array((result[colonne]))
            result[colonne] = np.around(a, decimals=5)

    return result
```

Nous avons fait le choix de remplacer les données de type NaN (Not A Number) par des 0 et d'arrondir les valeurs de type float à 5 chiffres après la virgule.

F. Conversion de l'adresse IP

Nous avons rencontré un problème avec deux colonnes de la dataset qui sont extrêmement importantes pour la suite du projet : 'SourceIP' et 'DestinationIP'. En effet ces colonnes sont de type 'IP' et ne sont donc pas interprétables par nos modèles. Nous avons donc effectué un traitement grâce à la fonction « IP2Int », pour pouvoir transformer des adresses IP en type Int tout en gardant une interprétation des adresses IP source et destination. Cette fonction permet d'enlever les points séparant les différents nombres et aussi d'associer à ces nombres des coefficients correspondant à leur place dans l'adresse.

Ces coefficients sont 2^{24} pour le premier indice, 2^{16} pour l'indice 2, 2^8 pour l'indice 3 et 2^0 pour le dernier indice.

Par exemple, si on prend l'adresse IP suivante : 192.168.20.191.

Notre algorithme va transformer cette adresse en un nombre entier :

$$2^{24} * 192 + 2^{16} * 168 + 2^8 * 20 + 2^0 * 191 = 3232240808$$

```
def IP2Int(df, column):
    i=0
    for values in df[column].values:
        o = list(map(int, values.split('.')))
        res = (16777216 * o[0]) + (65536 * o[1]) + (256 * o[2]) + o[3]
        df[column][i] = res
        i+=1
    return df
```

Ce résultat a deux avantages :

- Il est interprétable par nos modèles
- L'adresse IP est facilement retrouvable en effectuant l'opération inverse de notre fonction « IP2Int »

G. Normalisation des données

Certains modèles sont sensibles à un écart important entre deux valeurs extrêmes de certaines colonnes. Les résultats de nos modèles peuvent donc être impactés si nous ne réduisons pas l'écart entre nos valeurs minimales et maximales. Le but des fonctions « normalize » et « NormalizeDataset » est donc de réduire l'écart des valeurs de certaines colonnes entre 0 et 1.

Pour normaliser une valeur entre 0 et 1, nous effectuons l'opération suivante :

$$\text{Valeur} = \frac{\text{valeur} - \text{valeur minimale}}{\text{valeur maximale} - \text{valeur minimale}}$$

```
def normalize(colonne):
    max_value = colonne.max()
    min_value = colonne.min()
    colonne = np.round((colonne - min_value) / (max_value - min_value), 5)
    return colonne

def NormalizeDataset(dataset) :
    for colonne in dataset.columns:
        if(dataset[colonne].dtypes == 'float64'
           or dataset[colonne].dtypes == 'int64'
           and colonne != 'SourcePort'
           and colonne != 'DestinationPort'):

            dataset[colonne] = normalize(dataset[colonne])

    return dataset
```

Certaines colonnes ne doivent cependant pas être normalisées, car on peut perdre leur interprétation, comme les colonnes renseignant sur les ports source et destination. Nous les excluons alors de ce calcul.

V. Machine Learning

A. Théorie

Le machine learning constitue une manière de modéliser des phénomènes, dans le but de prendre des décisions stratégiques.

Le principe à partir de données brutes, il va sélectionner, nettoyer et transformer les données pertinentes. Par exemple, au lieu de récupérer directement le nombre de visites journalières, il va plutôt récupérer leurs variations d'un jour à l'autre.

Il faut ensuite explorer les données, c'est -à -dire visualiser les différentes variables, essayer de comprendre les comportements (valeurs extrêmes ou aberrantes, corrélation). Une fois qu'il a une bonne idée de ce qu'il a affaire, on peut définir une problématique plus précise sur laquelle répondre.

Plusieurs étapes sont à suivre avant de pouvoir utiliser une dataset :

- Récupération
- Nettoyage
- Exploration
- Modélisation
- Evaluation et interprétation
- Mise en production

Il faut ensuite modéliser notre système à partir des données qui nous ont été fournis. Modéliser signifie dans ce cas représenter le comportement d'un phénomène, afin de pouvoir aider à la résolution d'un problème concret de l'entreprise.

En machine learning , l'algorithme se construit sur une représentation interne afin de pouvoir effectuer la tâche qui lui est demandée (prédiction, identification). Il faut entrer un jeu de données afin que le système puisse s'entraîner et s'améliorer. Ce jeu de données s'appelle le training set. On peut appeler une entrée dans le jeu de données une instance ou une observation.

Voici la data du départ qui est un fichier de : avec une data Doh , une data non doh , une data malicieux et une data benin.

SourceIP, DestinationIP, SourcePort, DestinationPort, TimeStamp, Duration, FlowBytesSent, FlowSentRate, FlowBytesReceived, FlowReceivedRate, PacketLengthVariance,
192.168.20.191,176.103.130.131,50749,443,2020-01-14 15:49:11,95.081550,62311,655.3427031847924229253730087,65358,687.3888782839572977091770170,7,
192.168.20.191,176.103.130.131,50749,443,2020-01-14 15:50:52,122.309318,93828,767.1369731617667919626532461,101232,827.6720175972201888984451700,
192.168.20.191,176.103.130.131,50749,443,2020-01-14 15:52:55,120.958413,38784,320.6391274329963307306288815,38236,316.1086447124599758100331558,;
192.168.20.191,176.103.130.131,50749,443,2020-01-14 15:54:56,110.501080,61993,561.0171411899322612955457087,69757,631.2788979076041609729063282,;
176.103.130.131,192.168.20.191,443,50749,2020-01-14 15:56:46,54.229891,83641,1542.341289234750628578619124,76804,1416.266907119544090545931579,8,
192.168.20.191,176.103.130.131,52491,443,2020-01-14 15:57:40,145.460721,54084,371.8117140365336151468684113,63843,438.9019905930481397792604094,;
192.168.20.191,176.103.130.131,52742,443,2020-01-14 15:59:18,0.155410,1718,11054.62968920918859790232289,7411,47686.76404349784441155652789,2341,;
192.168.20.191,176.103.130.131,52742,443,2020-01-14 16:00:03,0.027001,55,2036.961594015036480130365542,66,2444.353912818043776156438650,30.25,5,5,;
192.168.20.191,176.103.130.131,52491,443,2020-01-14 16:00:06,44.649554,32285,723.0755317287155880661204365,36193,810.6016019779279318221185367,8,;
192.168.20.191,176.103.130.131,52742,443,2020-01-14 16:00:48,0.046455,55,1183.941448713809062533634700,66,1420.729738456570875040361640,30.25,5,5,;
192.168.20.191,176.103.130.131,52742,443,2020-01-14 16:01:33,44.890412,163,3.631064914262760609102897073,357,7.952700456391445015029044510,168.25,
192.168.20.191,176.103.130.131,52491,443,2020-01-14 16:01:35,91.338058,8090,88.57206050954137868795064594,7805,85.45178396501489006915386793,615,
192.168.20.191,176.103.130.131,52491,443,2020-01-14 16:03:17,121.332889,53598,441.7433759448355342466130515,59634,491.4908108715683840677361601,;
192.168.20.191,176.103.130.131,52491,443,2020-01-14 16:05:18,112.359885,56840,505.8744942645678215138792639,60236,536.0988043019089953678752875,;
176.103.130.131,192.168.20.191,443,52491,2020-01-14 16:07:10,135.244217,69610,514.6985323594279820482083903,65101,481.3588443489602220847638905,;
192.168.20.191,176.103.130.131,52491,443,2020-01-14 16:09:26,121.185931,35194,290.4132493729820832089824024,37434,308.8972431956643548003934549,;

Les données vont être nettoyées à l'aide d'une librairie qui se nomme panda, ce qui va être expliqué par la suite.

Nous allons entraîner notre algorithme, mais l'entraînement de la tâche sera appris à partir du train set



Un problème d'apprentissage comporte différents éléments spécifiques :

- Les données
- La tâche spécifique
- L'algorithme d'apprentissage
- L'analyse d'erreur (ou mesure des performances)

La base de données est un élément qui constitue la source principale de récupération de données.

Par exemple pour les données brutes qui souvent complexes et nécessitant des pré traitements spécifiques afin de le rendre manipulable par les algorithmes on pourra utiliser le deep Learning.

Les données peuvent être diverses, avec par exemple du texte simple, des images ou des vidéos. Les objets connectés sont aussi une grande source de données brutes qui récupèrent un grand nombre de données grâce à leurs capteurs.

La tâche spécifique à accomplir correspond au problème qu'on cherche à résoudre grâce à la modélisation du phénomène. On peut distinguer un certain nombre de cas qui reviennent souvent dans un environnement business, tels que les recommandations de produits par exemple. Je vous ai aussi déjà cité l'identification de transactions frauduleuses, la prédiction de l'impact d'une campagne marketing sur le taux de conversion ou la prédiction du prix optimal d'un produit pour maximiser le nombre de ventes. Un ordinateur apprend à partir des données pour résoudre une tâche en faisant attention à mesurer les performances.

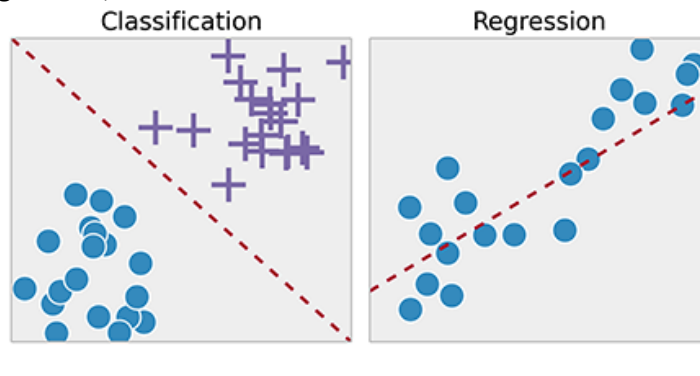
En apprentissage supervisé, on récupère des données dites annotées de leurs sorties pour entraîner le modèle, on associe un label ou une classe cible et on souhaite que l'algorithme soit capable une fois entraîné de prédire cette cible sur de nouvelles données non annotées.

Par exemple, si on souhaite classer des images, définir la catégorie de la photo. En apprentissage non supervisé, les données d'entrées ne sont pas annotées. L'algorithme d'entraînement s'applique dans ce cas à trouver les similitudes et les distinctions aux seins des

données et à les regrouper ensemble avec celles qui partagent des caractéristiques communes. Par exemple dans le cas des images seraient par exemple un moyen d'assembler automatiquement au sein d'une même catégorie.

Le choix d'un algorithme de machine learning dépend tout d'abord du type de donnée que nous avons.

Est-ce une valeur continue (un nombre) ou bien une valeur discrète (une catégorie). Le premier cas est appelé régression, le second une classification.



Par exemple, si on souhaite déterminer si l'image est un chat ou pas, on utilise une classification. En effet, ceci est binaire, l'image est un chat ou non. Tandis qu'une régression par exemple ce sera un nombre par exemple le coût d'un clic pour une page web.

B. Métriques

1. Matrice de confusion

		Classe réelle	
		-	+
Classe prédite	-	True Negatives (vrais négatifs)	False Negatives (faux négatifs)
	+	False Positives (faux positifs)	True Positives (vrais positifs)

$$\text{Nombre d'erreurs} = FP + FN$$

Les matrices de confusion permettent de pouvoir mesurer un modèle de classification et de pouvoir faire un bilan des données. Par exemple prenons l'exemple d'une femme enceinte, qui va voir un médecin pour savoir si elle est enceinte.

Les true negatives correspondent par exemple au fait que la femme ne pense pas être enceinte et que son médecin lui indique qu'elle n'est pas enceinte.

Un true positif correspond par exemple au fait que la femme pense être enceinte et effectivement elle est enceinte.

Un false positif correspond au fait que par exemple la femme pense ne pas être enceinte mais son médecin lui indique qu'elle est bien enceinte.

Un true false negative serait dans notre exemple, une femme qui pense être enceinte mais finalement elle ne l'est pas.

2. Rappel

On peut distinguer quelques valeurs qui seraient des mesures de performances.

Le rappel (« *recall* » en anglais), ou sensibilité (« *sensitivity* » en anglais), est le taux de vrais positifs, c'est-à-dire la proportion de positifs que l'on a correctement identifiés.

$$\text{Rappel} = \frac{TP}{TP + FN}$$

3. Précision

On s'intéressera donc aussi à la **précision**, c'est-à-dire la proportion de prédictions correctes parmi les points que l'on a prédits positifs

$$\text{Précision} = \frac{TP}{TP + FP}$$

4. F-measures

Pour évaluer un compromis entre rappel et précision on peut calculer le F-measures.

$$F - \text{mesure} = 2 \times \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}} = \frac{2TP}{2TP + FP + FN}$$

5. Spécificité

On s'intéresse aussi souvent à la spécificité ("*specificity*" en anglais), qui est le taux de vrais négatifs, autrement dit la capacité à détecter toutes les situations où la femme n'est pas enceinte.

$$\text{Spécificité} = \frac{TN}{FP + TN}$$

6. Accuracy

Accuracy : C'est la mesure de performance la plus intuitive. Il s'agit simplement du rapport entre le nombre d'observations correctement prédites et le nombre total d'observations. Cependant nous pouvons penser que si nous avons une précision élevée, notre modèle est le meilleur. Oui, la précision est une excellente mesure, mais seulement lorsque vous avez des ensembles de données symétriques où les valeurs des faux positifs et des faux négatifs sont presque identiques

$$\text{Accuracy} = (TP+TN)/(TP+FP+FN+TN)$$

7. F1 Score

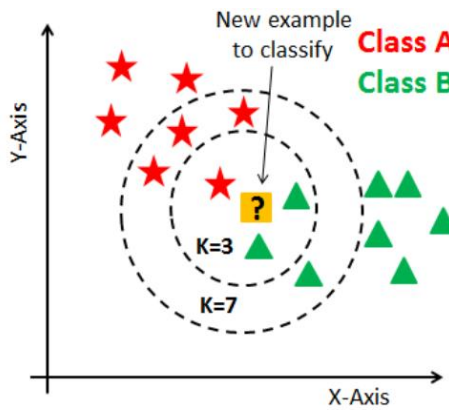
F1- Score : C'est la moyenne pondérée de Précision et Recall. Par conséquent, ce score tient compte à la fois des faux positifs et des faux négatifs. Intuitivement, il n'est pas aussi facile à comprendre que la précision, mais le score F1 est généralement plus utile que la précision, surtout si nous avons une distribution inégale des classes. La précision fonctionne mieux si les faux positifs et les faux négatifs ont un coût similaire. Si le coût des faux positifs et des faux négatifs est très différent, il est préférable d'examiner à la fois la précision et le rappel.

$$\text{F1 Score} = 2 * (\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

8. Courbe ROC

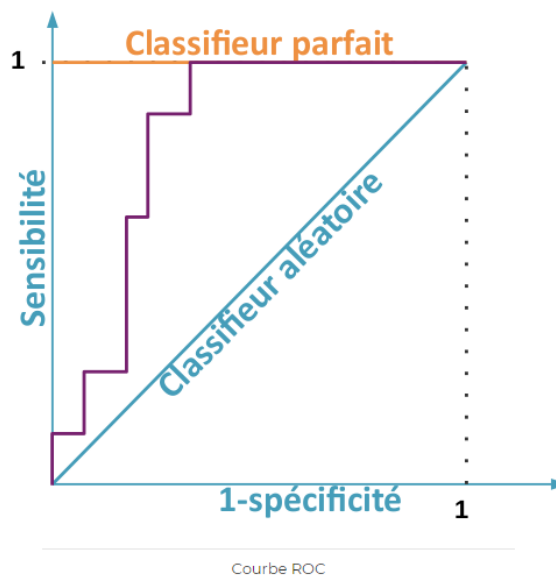
La courbe ROC " Receiver-Operator Characteristic " est une courbe pour montrer comment évolue la sensibilité évolue en fonction de la spécificité.

En bas à gauche de la courbe ROC, on prend la plus grande valeur prédite pour seuil, et on prédit donc que tous les points sont négatifs. En haut à droite, on prend la plus petite valeur prédite comme seuil, tous les points sont au-dessus et donc prédits positifs. Le reste de la courbe décrit toutes les situations intermédiaires.



Un modèle parfait va systématiquement associer des valeurs plus faibles aux exemples négatifs qu'aux exemples positifs. La courbe ROC correspondante dessine donc le coin supérieur gauche du carré.

Un modèle aléatoire, par contraste, va dessiner la diagonale du carré : quel que soit le seuil utilisé, comme le modèle est aléatoire, on aura la même proportion de prédictions positives correctes que de prédictions positives incorrectes. Le taux de faux positifs et le taux de vrais positifs seront donc égaux, et ainsi la sensibilité sera égale au complément à 1 de la sensibilité



On peut résumer la courbe ROC par un nombre : "l'aire sous la courbe", aussi dénotée AUROC pour *Area Under the ROC*, qui permet plus aisément de comparer plusieurs modèles.

Un classifieur parfait a une AUROC de 1 ; un classifieur aléatoire, une AUROC de 0.5.

Dans notre cas afin de choisir les meilleurs algorithmes, nous nous baserons sur différents critères :

- Precision
- F 1 Score
- Le temps
- Les matrices de confusion
- Accuracy

Nous avons réalisé des tests afin d'obtenir les meilleurs résultats en faisant varier les hyperparamètres. On peut réaliser une grille GridSearch, afin de d'obtenir les meilleurs

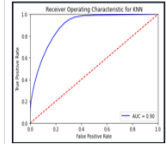
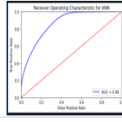
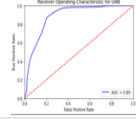
résultats. Par exemple pour les modèles KNN, on peut faire varier le nombre de voisins afin d'obtenir la meilleur Accuracy :

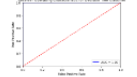
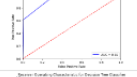
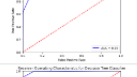
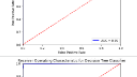
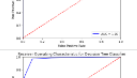
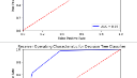
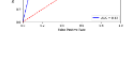
Meilleur(s) hyperparamètre(s) sur le jeu d'entraînement: {'n_neighbors': 7}

Résultats de la validation croisée :

accuracy = 0.749 (+/-0.015) for {'n_neighbors': 3}
accuracy = 0.752 (+/-0.020) for {'n_neighbors': 5}
accuracy = 0.757 (+/-0.024) for {'n_neighbors': 7}
accuracy = 0.754 (+/-0.016) for {'n_neighbors': 9}
accuracy = 0.755 (+/-0.021) for {'n_neighbors': 11}
accuracy = 0.755 (+/-0.021) for {'n_neighbors': 13}
accuracy = 0.757 (+/-0.025) for {'n_neighbors': 15}

Ce sont des paramètres que nous avons fait varier afin d'obtenir les meilleurs résultats.

KNN(DOH)(KNN)(DOH_n_neighbors=20)	0.86	0.94	14483.5s	Matrice de confusion : [[169279 18393] [28556 25200]]		0.94
KNN(DOH_n_neighbors=30)	0.83	0.89	9010.4s	Matrice de confusion : [[174247 5425] [36937 16819]]		0.97
GNB	0.8	0.88	2.16s	Matrice de confusion : [[175538 4134] [42927 10829]]		0.97

DECISION TREE CLASSIFIER - par défaut	0.9995	0.9995	0.9995	0.9994	[[49894 23] [26 49992]]	[[99.95, 0.05], [0.05, 99.95]]		21.94
DECISION TREE CLASSIFIER - changement max_depth = 2	0.7678	0.8549	0.8362	0.9643	[[35337 14580] [1784 48234]]	[[70.79, 29.21], [3.57, 96.43]]		4.27
DECISION TREE CLASSIFIER - changement max_depth = 3	0.9070	0.9367	0.9345	0.9681	[[44952 4965] [1580 48438]]	[[90.05, 9.95], [3.16, 96.84]]		5.32
DECISION TREE CLASSIFIER - changement max_depth = 4	0.9915	0.9465	0.9487	0.9053	[[49531 396] [4732 45286]]	[[99.23, 0.77], [9.46, 90.54]]		6.37
DECISION TREE CLASSIFIER - changement max_depth = 5	0.9903	0.9896	0.9898	0.9889	[[49436 481] [531 49457]]	[[99.04, 0.96], [1.06, 99.94]]		7.15
DECISION TREE CLASSIFIER - avec max_depth = 3 - changement criterion = 'entropy'	0.9094	0.9361	0.9341	0.9560	[[45113 4804] [1777 48241]]	[[90.38, 9.62], [3.55, 96.45]]		6.31
DECISION TREE CLASSIFIER - avec max_depth = 3 et criterion = 'entropy' changement du min_weight_fraction 0.0 -> 0.2	0.7206	0.8295	0.7990	0.9772	[[30673 18944] [1138 48890]]	[[82.05, 17.95], [2.28, 97.72]]		4.18

Le choix des meilleurs algorithmes se fera sur différents critères, il faut obtenir les meilleurs résultats, le fait qu'il n'y est pas de sur apprentissage et enfin que dans la matrice de confusion que le nombre de faux positive soit le plus élevé et que le nombre de faux négatif soit le moins élevé.

C. K-Nearest Neighbors

1. Explication du modèle

Les KNN 'k plus proches voisins' est une méthode non paramétrique dans laquelle le modèle mémorise les observations de l'ensemble d'apprentissage pour la classification des données de l'ensemble de test.

Pour prédire la classe d'une nouvelle donnée d'entrée, il va chercher ses K voisins les plus proches (en utilisant la distance euclidienne, ou autres) et choisira la classe des voisins majoritaires.

Le paramètre qu'on a décidé de modifier pour cet algorithme est **le nombre de voisin (k_neighbors)** ;

2. Mesure de performance

Concernant le modèle KNN , on a choisi le nombre de voisin avec lequel on a obtenu les meilleurs résultats qui était de 10 :

```
Report KNN
              precision    recall  f1-score   support

    0.0         0.96         0.71         0.82     179284
    1.0         0.77         0.97         0.86     179714

 accuracy              0.84     358998
 macro avg           0.87         0.84         0.84     358998
 weighted avg       0.87         0.84         0.84     358998

Temps pour KNN (en sec): 1880.0094
```

D. Gaussian Naives Bayes

GNB est un algorithme de classification pour les problèmes de classification binaires (à deux classes) et multiclassés. Elle est appelée "Bayes naïf" ou "Bayes idiot" parce que le calcul des probabilités pour chaque hypothèse est simplifié pour rendre leur calcul plus facile à comprendre. Le paramètre que nous faisons varier est le `var_smoothing`, qui correspond à la portion de la plus grande variance de toutes les caractéristiques qui est ajoutée aux variances pour la stabilité du calcul.

1. La classification Naïve Bayésienne

Tout d'abord, le théorème de Bayes décrit la probabilité d'une caractéristique en se basant au préalable sur des situations liées à celle-ci. Par exemple, si la probabilité qu'une personne soit diabétique est liée à son âge, alors en utilisant ce théorème, on peut utiliser l'âge pour prédire avec plus de précision la probabilité de souffrir de cette maladie chronique.

C'est un type de classification bayésienne probabiliste simple basée sur le théorème de Bayes avec une forte indépendance (dite naïve) des hypothèses. Elle met en œuvre un classifieur bayésien naïf, ou classifieur naïf de Bayes, appartenant à la famille des classifieurs linéaires.

En termes simples, un classifieur bayésien naïf suppose que l'existence d'une caractéristique pour une classe, est indépendante de l'existence d'autres caractéristiques. Un fruit peut être considéré comme une pomme s'il est rouge, arrondi, et fait une dizaine de centimètres. Même si ces caractéristiques sont liées dans la réalité, un classifieur bayésien naïf déterminera que le fruit est une pomme en considérant indépendamment ces caractéristiques de couleur, de forme et de taille.

Le mot naïf dans l'intitulé de l'algorithme implique que chaque paire de caractéristiques dans l'ensemble des données en question est indépendante de l'autre. C'est-à-dire que la valeur d'une caractéristique particulière est indépendante de la valeur de toute autre caractéristique pour une classe donnée.

Théorème de BAYE

$$p(C|F_1, \dots, F_n) = \frac{p(C) p(F_1, \dots, F_n|C)}{p(F_1, \dots, F_n)}.$$

Ou

$$\text{postérieure} = \frac{\text{antérieure} \times \text{vraisemblance}}{\text{évidence}}.$$

Avec C une variable de classe dépendante dont les instances ou *classes* sont peu nombreuses, conditionnée par plusieurs variables caractéristiques F_1, \dots, F_n .

A noter : Lorsque le nombre de caractéristiques n est grand, ou lorsque ces caractéristiques peuvent prendre un grand nombre de valeurs, baser ce modèle sur des tableaux de probabilités devient impossible.

$$\begin{aligned}
 p(C) p(F_1, \dots, F_n | C) \\
 &= p(C) p(F_1 | C) p(F_2, \dots, F_n | C, F_1) \\
 &= p(C) p(F_1 | C) p(F_2 | C, F_1) p(F_3, \dots, F_n | C, F_1, F_2) \\
 &= p(C) p(F_1 | C) p(F_2 | C, F_1) p(F_3 | C, F_1, F_2) p(F_4, \dots, F_n | C, F_1, F_2, F_3) \\
 &= p(C) p(F_1 | C) p(F_2 | C, F_1) p(F_3 | C, F_1, F_2) \cdots p(F_n | C, F_1, F_2, F_3, \dots, F_{n-1}).
 \end{aligned}$$

Nous faisons intervenir l'hypothèse naïve : si chaque F_i est indépendant des autres caractéristiques $F_{j \neq i}$

$$p(C | F_1, \dots, F_n) = \frac{1}{Z} p(C) \prod_{i=1}^n p(F_i | C)$$

Où Z (appelé « évidence ») est un facteur d'échelle qui dépend uniquement de F_1, \dots, F_n , à savoir une constante dans la mesure où les valeurs des variables caractéristiques sont connues.

Exemple d'application du classifieur : La classification des sexes

On cherche à classer chaque personne en tant qu'individu du sexe masculin ou féminin, selon les caractéristiques mesurées. Les caractéristiques comprennent la taille, le poids, et la pointure. On dispose d'un ensemble de données ;

Sexe	Taille (cm)	Poids (kg)	Pointure (cm)
masculin	182	81.6	30
masculin	180	86.2	28
masculin	170	77.1	30
masculin	180	74.8	25
féminin	152	45.4	15
féminin	168	68.0	20
féminin	165	59.0	18
féminin	175	68.0	23

La bibliothèque **Scikit-learn** de Python destinée à l'apprentissage automatique fournit le module **sklearn.naïve_bayes** qui contient plusieurs classificateurs Bayes Naïfs, dont chacun performe mieux sur un certain type de donnée.

Les types de classificateurs que la bibliothèque contient sont les suivants :

- 🚩 Gaussian Naïve Bayes ;
- 🚩 Multinomial Naïve Bayes ;
- 🚩 Complement Naïve Bayes ;
- 🚩 Bernoulli Naïve Bayes ;
- 🚩 Categorical Naïve Bayes.

2. Gaussian Naïve Bayes :

Il s'agit d'une variante de Bayes naïfs qui prend en charge les valeurs continues et part du principe que chaque classe est normalement distribuée. Afin de découvrir concrètement le fonctionnement de l'algorithme Gaussian Naïve Bayes, nous l'appliquerons sur notre Data set final intrusion de notre projet ;

```
@author: chant
"""

import pandas as pd
import time
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
data = pd.read_csv("df_total_csv_normalisee_Intrusion.csv", sep=';')

def GNB(df, colonne):
    print('Compilation algo gnb ...\n')
    t_debut = time.time()
    X = df.drop([colonne], axis = 1)
    y = df[colonne]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
    GNB = GaussianNB()
    # Create GNB
    GNB.fit(X_train, y_train)
    y_pred_GNB = GNB.predict(X_test)
    print('Report GNB \n', classification_report(y_test, y_pred_GNB))
    t_fin = time.time()
    t_total = t_fin - t_debut
    print("Temps pour GNB (en sec): ", t_total)
    return GNB
GNB(data, 'Intrusion')
```

Et voici le résultat :

```
Report GNB
              precision    recall  f1-score   support

     0.0         0.00         0.00         0.00         3885
     1.0         0.93         1.00         0.96        50044

 accuracy              0.93         53929
 macro avg              0.46         0.50         0.48         53929
weighted avg              0.86         0.93         0.89         53929

Temps pour GNB (en sec): 0.576998233795166
C:\Users\chant\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1272:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

3. Bernoulli Naïve Bayes

En guise de découverte de l'algorithme Bernoulli Naïve Bayes, nous l'appliquerons sur notre Data set final intrusion de notre projet

```
@author: chant
"""

import pandas as pd
import time
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
data = pd.read_csv("df_total_csv_normalisee_Intrusion.csv", sep=';')
print(data)
```

En résultat, nous affichons l'ensemble de nos données ;

```
In [9]: runfile('C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi/test2.py', wdir='C:/
Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi')

SourceIP ... Intrusion
0      1.00000 ...      1.0
1      0.03667 ...      1.0
2      0.03667 ...      1.0
3      0.03667 ...      1.0
4      0.03667 ...      1.0
...      ... ...      ...
269638  0.00000 ...      0.0
269639  0.00000 ...      0.0
269640  0.00000 ...      0.0
269641  0.00000 ...      0.0
269642  0.00000 ...      0.0
```

Le Naïve Bayes Classifier peut être appliqué dans différents scénarios, un des cas d'utilisation classiques pour ce modèle d'apprentissage est la classification des documents. Il s'agit de déterminer si un document correspond à certaines catégories ou pas

4. Mesure de performance

Concernant le modèle GNB, on a fait varier le `var_smoothing` le meilleur résultat était :

Training GNB ...

Report GNB

	precision	recall	f1-score	support
0.0	0.54	0.98	0.70	179136
1.0	0.89	0.18	0.30	179862
accuracy			0.58	358998
macro avg	0.72	0.58	0.50	358998
weighted avg	0.72	0.58	0.50	358998

Temps pour GNB (en sec): 30.3485

E. Support Vector Machine

1. Explication du modèle

Les SVM sont une famille d'algorithmes d'apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression ou de détection d'anomalie. Ils sont connus pour leurs solides garanties théoriques, leur grande flexibilité ainsi que leur simplicité d'utilisation même sans grande connaissance de data mining.

Leur principe est simple : ils ont pour but de séparer les données en classes à l'aide d'une frontière aussi « simple » que possible, de telle façon que la distance entre les différents groupes de données et la frontière qui les sépare soit maximale. Cette distance est aussi appelée « marge » et les SVMs sont ainsi qualifiés de séparateurs à vaste marge, les vecteurs de support étant les données les plus proches de la frontière.

2. Mesure de performance

Concernant le modèle SVM, il est très précis avec les résultats qu'on a obtenu au premier semestre :

Report Support Vector Machine					
	precision	recall	f1-score	support	
	0.0	0.99	0.99	0.99	179672
	1.0	0.98	0.97	0.97	53756
accuracy				0.99	233428
macro avg		0.98	0.98	0.98	233428
weighted avg		0.99	0.99	0.99	233428
Temps pour SVM classifieur (en sec): 6778.447428703308					

Cependant il prend un temps énorme, l'exécution peut durer des heures. Il est très précis malgré tout.

F. Perceptron

1. Explication du modèle

Le **Perceptron** est un algorithme d'apprentissage automatique linéaire pour les tâches de classification binaire.

Il peut être considéré comme l'un des premiers et l'un des types les plus simples de réseaux de neurones artificiels. Il ne s'agit certainement pas d'un apprentissage «profond», mais d'un élément de base important.

C'est un type de modèle de réseau neuronal, peut-être le type le plus simple de modèle de réseau neuronal.

Il se compose d'un seul nœud ou neurone qui prend une ligne de données en entrée et prédit une étiquette de classe. Ceci est réalisé en calculant la somme pondérée des entrées et un biais (mis à 1). La somme pondérée de l'entrée du modèle est appelée l'activation.

Le paramètre qu'on a changé est également le `class_weight`.

Activation = poids * entrées + biais

Si l'activation est supérieure à 0,0, le modèle affichera 1,0; sinon, il affichera 0.0.

✚ **Prédiction 1** : Si Activation > 0,0

✚ **Prédiction 0** : Si Activation <= 0,0

2. Implémentation du modèle

Nous utiliserons la fonction `make_classification()` pour créer un ensemble de données avec 1000 exemples, chacun avec 20 variables d'entrée.

```
@author: chant
"""

from sklearn.datasets import make_classification
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_r
# summarize the dataset
print(X.shape, y.shape)
```

L'exécution de l'exemple crée l'ensemble de données et confirme le nombre de lignes et de colonnes de l'ensemble de données.

```
In [9]: runfile('C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi/perceptron.py',
wdir='C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi')
(1000, 10) (1000,)

In [10]:
```

Nous pouvons ajuster et évaluer un modèle Perceptron en utilisant la validation croisée de la classe `RepeatedStratifiedKFold`. Nous utiliserons 10 plis et trois répétitions dans le harnais de test.

```
# define model
model = Perceptron()
```

L'exemple complet d'évaluation du modèle Perceptron pour la tâche de classification binaire synthétique est présenté ci-dessous.

```
@author: chant
"""

# evaluate a perceptron model on the dataset
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_r
# define model
model = Perceptron()
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# summarize result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

L'exécution de l'exemple évalue l'algorithme Perceptron sur l'ensemble de données synthétiques et indique la précision moyenne sur les trois répétitions de la validation croisée par 10.

Les résultats spécifiques peuvent varier compte tenu de la nature stochastique de l'algorithme d'apprentissage. Il est souhaitable d'exécuter l'exemple plusieurs fois.

Dans ce cas, nous pouvons voir que le modèle a atteint une précision moyenne d'environ 84,7%.

```
In [13]: runfile('C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi/perceptron1.py',
wdir='C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi')
Mean Accuracy: 0.847 (0.052)
```

Nous pouvons décider d'utiliser le classificateur Perceptron comme modèle final et faire des prédictions sur de nouvelles données.

Cela peut être réalisé en ajustant le pipeline de modèle sur toutes les données disponibles et en appelant la fonction predict () en passant une nouvelle ligne de données.

Nous pouvons le démontrer avec un exemple complet ci-dessous.

```

# make a prediction with a perceptron model on the dataset
from sklearn.datasets import make_classification
from sklearn.linear_model import Perceptron
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_r
# define model
model = Perceptron()
# fit model
model.fit(X, y)
# define new data
row = [0.12777556, -3.64400522, -2.23268854, -1.82114386, 1.75466361, 0.1243966, 1.03
# make a prediction
yhat = model.predict([row])
# summarize prediction
print('Predicted Class: %d' % yhat)

```

L'exécution de l'exemple correspond au modèle et effectue une prédiction d'étiquette de classe pour une nouvelle ligne de données

```

In [15]: runfile('C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi/perceptron2.py',
wdir='C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi')
Predicted Class: 1

```

3. Configuration des hyperparamètres

L'hyperparamètre le plus important est peut-être le taux d'apprentissage.

Un taux d'apprentissage élevé peut amener le modèle à apprendre rapidement, mais peut-être au prix d'une moindre compétence. Un taux d'apprentissage plus faible peut aboutir à un modèle plus performant, mais peut prendre beaucoup de temps pour entraîner le modèle.

Aussi il est courant de tester les taux d'apprentissage sur une échelle logarithmique entre une petite valeur telle que $1e-4$ (ou inférieure) et 1,0. Nous allons tester les valeurs suivantes dans ce cas :

```
# define grid
grid = dict()
grid['eta0'] = [0.0001, 0.001, 0.01, 0.1, 1.0]
```

L'exemple ci-dessous illustre cela en utilisant la classe GridSearchCV avec une grille de valeurs que nous avons définies.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Feb  4 12:20:23 2021
4
5  @author: chant
6  """
7
8
9  # grid search learning rate for the perceptron
10 from sklearn.datasets import make_classification
11 from sklearn.model_selection import GridSearchCV
12 from sklearn.model_selection import RepeatedStratifiedKFold
13 from sklearn.linear_model import Perceptron
14 # define dataset
15 X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_r
16 # define model
17 model = Perceptron()
18 # define model evaluation method
19 cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
20 # define grid
21 grid = dict()
22 grid['eta0'] = [0.0001, 0.001, 0.01, 0.1, 1.0]
23 # define search
24 search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)
25 # perform the search
26 results = search.fit(X, y)
27 # summarize
28 print('Mean Accuracy: %.3f' % results.best_score_)
29 print('Config: %s' % results.best_params_)
30 # summarize all
31 means = results.cv_results_['mean_test_score']
32 params = results.cv_results_['params']
33 for mean, param in zip(means, params):
34     print(">%.3f with: %r" % (mean, param))
```

L'exécution de l'exemple évaluera chaque combinaison de configurations en utilisant une validation croisée répétée.

Les résultats spécifiques peuvent varier compte tenu de la nature stochastique de l'algorithme d'apprentissage.

Dans ce cas, nous pouvons voir qu'un taux d'apprentissage plus petit que le taux par défaut se traduit par de meilleures performances avec un taux d'apprentissage de 0,0001 et 0,001 tous deux atteignant une précision de classification d'environ 85,7% par rapport à la valeur par défaut de 1,0 qui a atteint une précision d'environ 84,7%.

```
In [16]: runfile('C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi/perceptron3.py',
wdir='C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi')
Mean Accuracy: 0.857
Config: {'eta0': 0.0001}
>0.857 with: {'eta0': 0.0001}
>0.857 with: {'eta0': 0.001}
>0.853 with: {'eta0': 0.01}
>0.847 with: {'eta0': 0.1}
>0.847 with: {'eta0': 1.0}
```

Un autre hyperparamètre important est le nombre d'époques utilisées pour entraîner le modèle.

Cela peut dépendre du jeu de données d'entraînement et peut varier considérablement. Encore une fois, nous explorerons les valeurs de configuration sur une échelle logarithmique entre 1 et $1e + 4$.

```
# define grid
grid = dict()
grid['max_iter'] = [1, 10, 100, 1000, 10000]
# define search
```

Nous utiliserons notre taux d'apprentissage performant de 0,0001 trouvé dans la recherche précédente.

```
# define model
model = Perceptron(eta0=0.0001)
```

L'exemple complet de grille(grid) recherchant le nombre d'époques d'entraînement est listé ci-dessous.

```

# grid search total epochs for the perceptron
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_r
# define model
model = Perceptron(eta0=0.0001)
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['max_iter'] = [1, 10, 100, 1000, 10000]
# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X, y)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
# summarize all
means = results.cv_results_['mean_test_score']
params = results.cv_results_['params']
for mean, param in zip(means, params):
    print(">%.3f with: %r" % (mean, param))

```

Nous pouvons voir que les époques 10 à 10 000 donnent à peu près la même précision de classification. Une exception intéressante serait d'explorer la configuration du taux d'apprentissage et du nombre d'époques de formation en même temps pour voir si de meilleurs résultats peuvent être obtenus.

```

In [17]: runfile('C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi/perceptron4.py',
wdir='C:/Users/chant/OneDrive/Documents/ITS2/semestre3/Mespi')
Mean Accuracy: 0.857
Config: {'max_iter': 10}
>0.850 with: {'max_iter': 1}
>0.857 with: {'max_iter': 10}
>0.857 with: {'max_iter': 100}
>0.857 with: {'max_iter': 1000}
>0.857 with: {'max_iter': 10000}

```

4. Mesure de performance

Par exemple pour l'algorithme Perceptron , avec des hyper paramètres par défaut :

```
Training Perceptron ...

Report Perceptron

              precision    recall  f1-score   support

     0.0         0.00      0.00      0.00     179136
     1.0         0.50      1.00      0.67     179862

 accuracy          0.50      358998
 macro avg         0.25      0.50      0.33      358998
weighted avg         0.25      0.50      0.33      358998

Temps pour Perceptron (en sec): 29.0513

Matrice de confusion :
[[    0 179136]
 [    0 179862]]

Accuracy : 0.5010111476944161

Precision : 0.5010111476944161

F1 score : 0.6675648591470883
```

Lorsque que on change l'hyperparamètres : classWeight on obtient :

```
Training Perceptron ...

Report Perceptron

              precision    recall  f1-score   support

     0.0         0.54      0.98      0.70     179136
     1.0         0.89      0.18      0.30     179862

 accuracy          0.58      358998
 macro avg         0.72      0.58      0.50      358998
weighted avg         0.72      0.58      0.50      358998

Temps pour Perceptron (en sec): 31.4778

Matrice de confusion :
[[175124   4012]
 [147151  32711]]

Accuracy : 0.5789308018429072

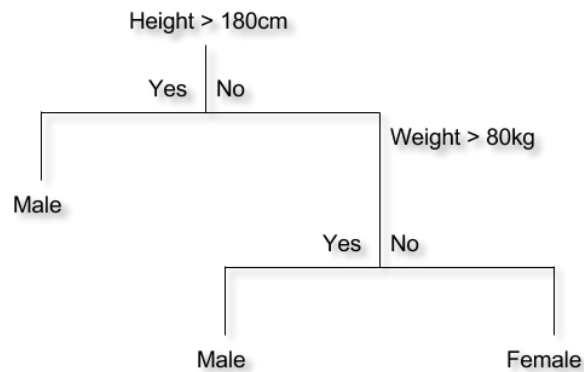
Precision : 0.8907496664215887

F1 score : 0.3020615462751345
```

G. Decision Tree Classifier

1. Explication du modèle

Un des modèles que nous avons choisis est le Decision Tree Classifier. Ce modèle se construit en posant une série de questions sur l'ensemble des données, à chaque fois qu'une réponse est donnée, une nouvelle question est posée, jusqu'à ce que l'on obtienne une étiquette pour nos données. Nous pouvons voir un exemple pour comprendre le principe :



Donc, une série de questions est posée pour savoir si les données correspondent à une Femme ou un Homme, en fonction de différentes caractéristiques.

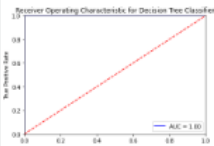
Le Decision Tree Classifier que nous utilisons dans notre projet suit donc le même principe, il va parcourir une série de questions (les features de notre dataset) pour déterminer à la fin si les données qui lui sont présentées correspondent à un DoH ou non DoH.

2. Optimisation du modèle

Afin d'instancier un bon modèle, nous avons fait varier certains paramètres du Decision Tree Classifier, et comparer les métriques suivantes : ROC Curve, Precision, Rappel, F1 Score, Accuracy, Matrice de confusion et le temps d'entraînement.

a) Hyperparamètres par défaut

Tout d'abord, nous avons entraîné le modèle avec ses paramètres par défaut.

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9994	0.9995	0.9995	0.9994	[[179610 104] [64 179220]]	[[99.94, 0.06], [0.04, 100.0]]		95.13

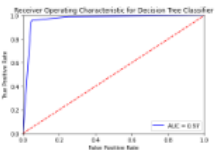
Nous nous sommes rendu compte que le modèle sur-apprenait, toutes les métriques confirment cela.

b) Changement de max_depth

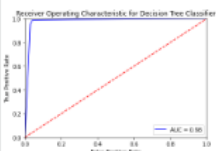
Nous avons donc décidé de modifier le maximum de la profondeur de l'arbre (max_depth), c'est-à-dire qu'il s'agit d'arrêter le développement de l'arbre une fois qu'il a atteint une certaine profondeur, cela évite donc que l'arbre construise des branches avec peu d'exemples et permet d'éviter le sur-apprentissage.

Par défaut, cet hyperparamètre est à None, nous l'avons donc fait varier à 2, 3, 4 et 5. Voici les résultats pour chaque profondeur :

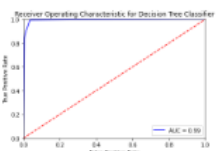
A max_depth = 2 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9504	0.9545	0.9544	0.9476	[[170749 8965] [7393 171891]]	[[95.01, 4.99], [4.12, 95.91]]		31.46

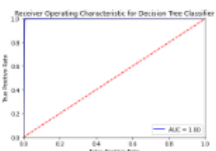
A max_depth = 3 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9594	0.9734	0.9731	0.9578	[[172225 7489] [2166 177118]]	[[95.83, 4.17], [1.21, 98.82]]		34.90

A max_depth = 4 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9672	0.9799	0.9797	0.9650	[[173684 6030] [1241 178043]]	[[96.64, 3.36], [0.69, 99.34]]		38.807

A max_depth = 5 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9934	0.9934	0.9935	0.9935	[[178535 1179] [1154 178130]]	[[99.34, 0.66], [0.64, 99.39]]		41.67

Si l'on prend les résultats de la précision, F1 Score, Accuracy, Recall et même la matrice de confusion le meilleur semble être le Decision Tree avec une profondeur de 5, mais sa courbe ROC nous montre que ce dernier sur-apprend, puisqu'elle est très en haut, nous éliminons donc le max_depth = 5.

Nous avons pour la profondeur de 4, la même argumentation, le modèle est bon mais peut être parfait en regardant sa courbe ROC, notre choix se portera donc entre max_depth = 2 ou 3.

Les deux modèles se valent, le Decision Tree avec une profondeur de 2 est certes plus rapide, mais moins précis (même si les métriques, comme la precision, accuracy, rappel, f1 score et courbe ROC sont très proches pour les deux modèles). Notre choix s'est tourné vers le modèle avec une profondeur de 3, parce que lorsqu'on regarde la matrice de confusion il y a nettement moins de Faux Négatif que pour le modèle avec max_depth = 2 avec 4.12% contre 1.21%. Nous pensons qu'il est plus grave que le modèle prédise un négatif qui n'en est pas un, puisqu'il

déterminera que les données représentent un non-DoH, et passera donc à côté d'une possible intrusion sur un DoH.

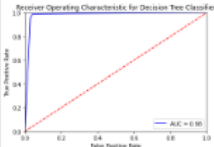
c) *Changement du critère*

Le deuxième paramètre que nous avons modifié est le critère de l'arbre. Cet hyperparamètre va permettre de tester et évaluer la qualité de toutes les nouvelles décisions qu'il est possible d'ajouter à l'arbre, soit en calculant le score Gini soit l'entropie. Par défaut, le critère est à Gini, ce score est compris entre 0 et 0.5, qui indique la probabilité que l'arbre se trompe lors d'une prise de décision, il se calcule grâce à la formule suivante, on appelle également ce score l'impureté Gini :

$$GiniIndex = 1 - \sum_j p_j^2$$

Où p_j est la probabilité d'obtenir la classe j .

Pour comparer par la suite, voici de nouveau les métriques obtenues avec ce paramètre (et max_depth = 3) :

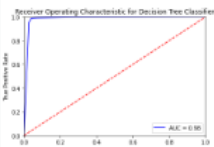
Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9594	0.9734	0.9731	0.9578	[[172225 7489] [2166 177118]]	[[95.83, 4.17], [1.21, 98.82]]		34.90

Nous avons donc essayé l'entropie comme critère d'évaluation de la qualité, qui indique l'incohérence des caractéristiques avec la variable cible, ce score est très semblable au score Gini mais est compris entre 0 et 1 et se calcule par la formule suivante :

$$Entropy = - \sum_j p_j \cdot \log_2 \cdot p_j$$

Où p_j est la probabilité d'obtenir la classe j .

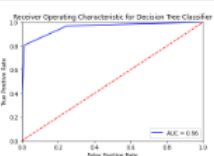
Nous avons donc instancié le Decision Tree Classifier avec ce critère et voici les métriques obtenues :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9611	0.9737	0.9734	0.9607	[[172573 7141] [2385 176899]]	[[96.03, 3.97], [1.33, 98.7]]		35.21

Bien que l'entropie soit plus complexe niveau informatique, car le calcul utilise des logarithmes, le modèle avec ce critère est tout de même un petit peu plus rapide. Les autres métriques sont quasiment identiques nous avons décidé d'utiliser le critère de l'entropie pour notre modèle.

d) Changement du `min_weight_fraction`

Pour finir nous avons testé l'hyperparamètre `min_weight_fraction_leaf`, qui par défaut est à 0.0, à 0.2. Ce paramètre est la fraction des échantillons d'entrée nécessaires pour être au niveau d'un nœud feuille.

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9853	0.8868	0.8972	0.9883	[[177569 2145] [34748 144536]]	[[98.81, 1.19], [19.39, 80.64]]		29.96

Nous constatons que la courbe ROC est nettement moins bonne, le modèle se trompe énormément et détecte des faux négatifs. Alors même si ce modèle est plus précis, nous avons décidé de garder la valeur par défaut pour ce paramètre.

3. Implémentation du modèle

Nous avons donc instancié un Decision Tree Classifier avec une profondeur maximale de l'arbre de 3, un critère d'évaluation de la qualité des scissions basée sur l'entropie, et les autres hyperparamètres par défaut.

```
DTC = DecisionTreeClassifier(criterion='entropy', splitter='best', max_depth=3, min_samples_split=2,
                             min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
                             max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None,
                             ccp_alpha=0.0)
```

Nous l'avons entraîné sur les données d'entraînement : `DTC = DTC.fit(X_train,y_train)`

Les données d'entraînement représentent 80% du dataset que nous avons au préalable équilibré avec l'algorithme SMOTE (expliqué précédemment dans le processing des données) comme suit :

```
oversample = SMOTE()
X_smote, y_smote = oversample.fit_sample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)
```

Nous avons également créé une fonction qui permet de retourner la prédiction de ce modèle sur de nouvelles données et qu'on utilisera à la fin dans l'interface graphique (quand l'utilisateur insère donc de nouvelles données).

```
def DTC_Prediction(df, DTC):
    X_new_DTC = df
    # Predict
    new_prediction_DTC = DTC.predict(X_new_DTC)
    print("New prediction DTC: {}".format(new_prediction_DTC))
    return new_prediction_DTC
```

Cette fonction sera appelée avec le modèle pré-entraîné et sauvegardé grâce à la librairie pickle.

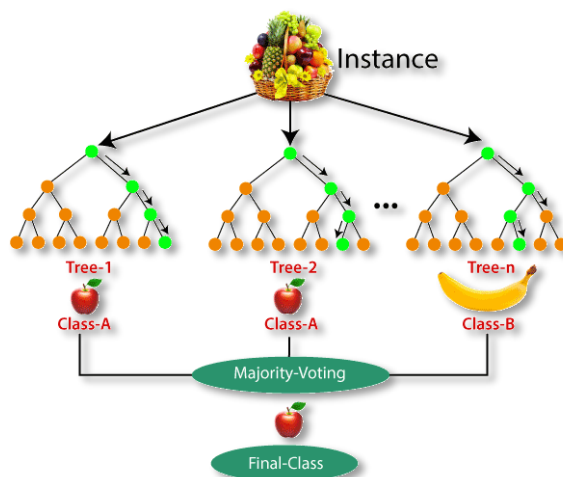
```
# DTC Decision Tree Classifier
DTC_DoH = ml.DTC(df_total_csv_normalisee_DoH, colonne)
filename_DTC_DoH = 'DoH/finalized_model_DTC_DoH.sav'
pickle.dump(DTC_DoH, open(filename_DTC_DoH, 'wb'))
```

H. Random Forest Classifier

1. Explication du modèle

Le modèle suivant choisi est le Random Forest Classifier, qui permet de construire plusieurs arbres de décision et de tous les entraîner. Chaque arbre donnera un résultat, et la conclusion du Random Forest Classifier sera le résultat qui revient le plus souvent, c'est le vote majoritaire.

Comme nous pouvons le voir sur l'exemple ci-dessous, chaque arbre a un résultat, deux concluent que le fruit est une pomme et un que le fruit est une banane, alors le Random Forest Classifier donnera comme résultat que le fruit donné est une pomme.



Le Random Forest que nous instancions dans notre projet fonctionne comme cela, et déterminera donc par la majorité s'il est question d'un DoH ou d'un non DoH.

2. Optimisation du modèle

Afin d'instancier un bon modèle, nous avons fait varier certains paramètres du Random Forest Classifier, et comparer les métriques suivantes : ROC Curve, Precision, Rappel, F1 Score, Accuracy, Matrice de confusion et le temps d'entraînement.

a) Hyperparamètres par défaut

Dans un premier temps, nous avons entraîné le modèle avec ses paramètres par défaut.

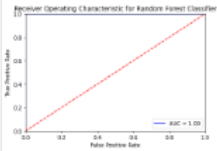
Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9995	0.9997	0.9997	0.9994	[[179786 76] [13 179123]]	[[99.96, 0.04], [0.01, 99.99]]		771.60

Comme nous avons pu le constater avec le Decision Tree Classifier, ce modèle sur-apprend avec ses paramètres par défaut. Par exemple sa courbe ROC est bien trop parfaite, ainsi que les autres métriques. De plus, ce modèle est bien plus long, ce qui s'explique par le fait qu'il possède plusieurs arbres.

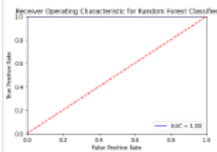
b) Changement du $n_estimators$

Nous avons décidé de tester différentes possibilités pour le paramètre $n_estimators$ qui est le nombre d'arbres que le Random Forest possède avant d'effectuer le vote majoritaire, par défaut il a 100 arbres, nous avons donc testé avec 50 puis 200. Voici les résultats obtenus :

Pour $n_estimators = 50$:

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9994	0.9996	0.9996	0.9995	[[179769 93] [24 179112]]	[[99.95, 0.05], [0.01, 99.99]]		413.01

Pour $n_estimators = 200$:

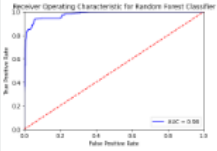
Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9994	0.9996	0.9996	0.9995	[[179771 91] [20 179116]]	[[99.95, 0.05], [0.01, 99.99]]		1533.92

Comme on peut le constater grâce à ces deux tableaux, ce changement ne modifie en rien les métriques de notre modèle, nous avons donc décidé de garder l'hyperparamètre par défaut à 100.

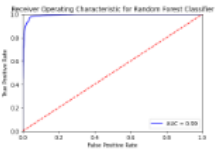
c) Changement de max_depth

Ensuite nous avons fait varier l'hyperparamètre max_depth qui est défini comme le chemin le plus long entre le nœud racine et le nœud feuille dans un arbre du Random Forest. Par défaut, ce paramètre n'a pas de valeur, nous voulions donc limiter la profondeur à laquelle chaque arbre grandisse. Nous avons testé ce paramètre à 1, 2, 3 et 5, voici les tableaux résumant les différentes métriques étudiées :

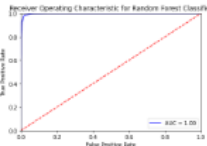
A $max_depth = 1$:

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9439	0.9314	0.9324	0.9307	[[170091 9771] [14470 164666]]	[[94.57, 5.43], [8.08, 91.92]]		85.13

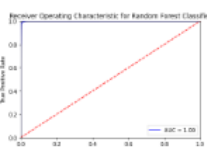
A $max_depth = 2$:

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9622	0.9637	0.9637	0.9644	[[173072 6790] [6216 172920]]	[[96.22, 3.78], [3.47, 96.53]]		143.79

A $max_depth = 3$:

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9778	0.9819	0.9819	0.9711	[[175863 3999] [2487 176649]]	[[97.78, 2.22], [1.39, 98.61]]		172.03

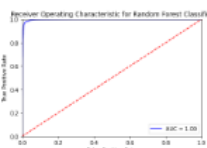
A max_depth = 5 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9881	0.9905	0.9905	0.9901	[[177736 2126] [1253 177883]]	[[98.82, 1.18], [0.7, 99.3]]		252.84

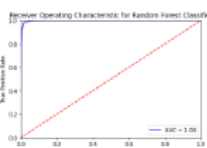
Pour commencer, nous constatons assez vite que le modèle avec max_depth = 5 sur-apprend, toutes ses métriques le confirment. Ensuite si nous comparons les trois autres modèles, que le modèle qui se trompe moins que les autres, mais qui est également plus précis, est celui avec une profondeur d'arbre à 3. Nous avons donc décidé de choisir cette valeur pour l'hyperparamètre max_depth.

d) *Changement de class_weight*

Pour finir, nous avons décidé de modifier l'hyperparamètre class_weight, qui par défaut est à None, et nous l'avons essayé en 'balanced'. Ce paramètre correspond aux poids associés à chaque classe dans les arbres. Lorsqu'il est sur 'balanced', le modèle utilise les valeurs de la cible 'y' pour ajuster automatiquement les poids qui sont inversement proportionnels aux fréquences des classes dans les données d'entrée.

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9680	0.9758	0.9756	0.9772	[[174042 5820] [2920 176216]]	[[96.76, 3.24], [1.63, 98.37]]		192.53

Si l'on compare ces résultats à ceux qui suivent (qui sont avec max_depth = 3) :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9778	0.9819	0.9819	0.9711	[[175863 3999] [2487 176649]]	[[97.78, 2.22], [1.39, 98.61]]		172.03

Nous remarquons que le modèle avec ce class_weight est légèrement moins précis et plus long, nous avons donc décidé de laisser None pour ce paramètre.

3. Implémentation du modèle

Nous avons donc instancié un Random Forest Classifier avec une profondeur des arbres à 3 et les autres hyperparamètres par défaut.

```
RFC = RandomForestClassifier(n_estimators=100, criterion='gini',
                             max_depth=3, min_samples_split=2, min_samples_leaf=1,
                             min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
                             min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True,
                             oob_score=False, n_jobs=None, random_state=None, warm_start=False,
                             class_weight=None, ccp_alpha=0.0, max_samples=None)
```

Nous l'avons entraîné sur les données d'entraînement : `RFC = RFC.fit(X_train,y_train)`

Les données d'entraînement représentent 80% du dataset que nous avons au préalable équilibré avec l'algorithme SMOTE comme suit :

```
oversample = SMOTE()
X_smote, y_smote = oversample.fit_sample(X, y)

X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)
```

Comme pour le Decision Tree Classifier, nous avons également créé une fonction qui permet de retourner la prédiction de ce modèle sur de nouvelles données et qu'on utilisera à la fin dans l'interface graphique (quand l'utilisateur insère donc de nouvelles données).

```
def RFC_Prediction(df, RFC):
    X_new_RFC = df
    # Predict
    new_prediction_RFC = RFC.predict(X_new_RFC)
    print("New prediction RFC: {}".format(new_prediction_RFC))
    return new_prediction_RFC
```

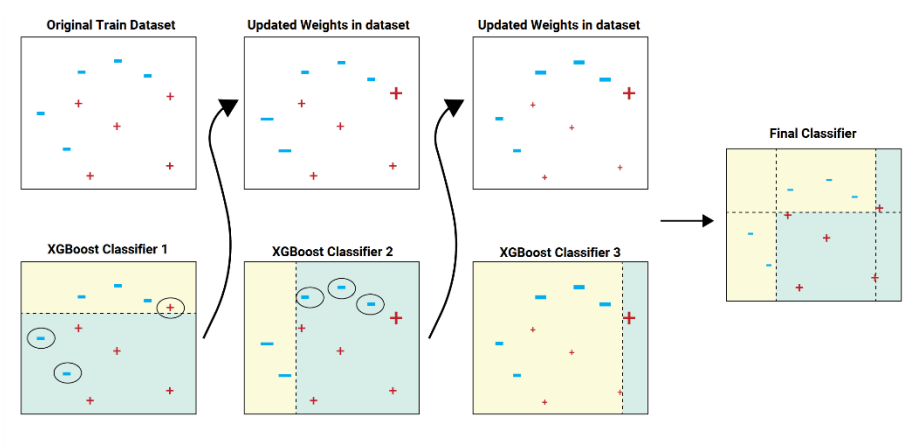
Cette fonction sera appelée avec le modèle pré-entraîné et sauvegardé grâce à la librairie pickle.

```
# RFC Random Forest Classifier
RFC_DoH = ml.RFC(df_total_csv_normalisee_DoH, colonne)
filename_RFC_DoH = 'DoH/finalized_model_RFC_DoH.sav'
pickle.dump(RFC_DoH, open(filename_RFC_DoH, 'wb'))
```


I. XGBoost Classifier

1. Explication du modèle

Pour le dernier modèle, nous avons décidé d'utiliser le XGBoost (eXtreme Gradient Boosting), qui assemble plusieurs modèles de Machine Learning peu performants pour en créer un plus efficace et satisfaisant. Le but est donc de créer séquentiellement des arbres faibles afin que chaque nouvel arbre se concentre sur la faiblesse du précédent en donnant des poids aux données là où le modèle se trompe, et donner finalement un bon modèle.



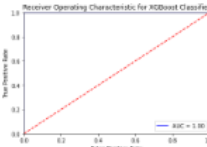
Comme nous pouvons le voir sur l'image ci-dessus, le premier modèle donne un résultat en séparant les plus et les moins, mais se trompe sur certains. Alors ces erreurs se verront attribuer un poids plus important que les autres, donc le deuxième modèle prendra en compte ces erreurs et se concentrera dessus pour avoir finalement un modèle qui ne se trompera pas sur la séparation des données.

2. Optimisation du modèle

Afin d'instancier un bon modèle, nous avons fait varier certains paramètres du Random Forest Classifier, et comparer les métriques suivantes : ROC Curve, Precision, Rappel, F1 Score, Accuracy, Matrice de confusion et le temps d'entraînement.

a) Hyperparamètres par défaut

Dans un premier temps, nous avons entraîné le modèle avec ses paramètres par défaut.

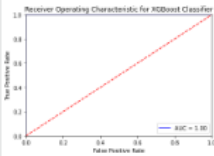
Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9999	0.9999	0.9999	0.9999	[[179847 15] [5 179131]]	[[99.99, 0.01], [0.0, 100.0]]		473.44

Ce modèle est plus qu'efficace, et nous pensons même à du sur-apprentissage, nous avons donc cherché à modifier certains hyperparamètres.

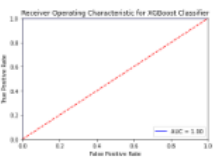
b) Changement de max_depth

Nous avons voulu faire varier le max_depth qui est la profondeur maximale d'un arbre, et qu'on utilise pour contrôler le sur-apprentissage, ce qui est parfait dans notre cas. Par défaut ce paramètre est à 6, nous avons donc testé 7 et 8. Voici les résultats obtenus :

A max_depth = 7 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9999	0.9999	0.9999	0.9999	[[179853 9] [6 179130]]	[[99.99, 0.01], [0.0, 100.0]]		590.96

A max_depth = 8 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9999	0.9999	0.9999	0.9999	[[179849 13] [5 179131]]	[[99.99, 0.01], [0.0, 100.0]]		646.56

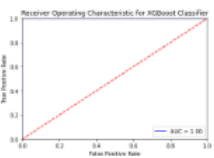
Comme on peut le voir, il n'y a eu aucun changement, on a donc décidé de garder ce paramètre par défaut.

c) Changement du learning_rate

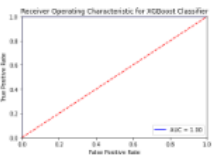
Nous avons essayé de modifier le learning_rate, qui est donc le taux d'apprentissage et qui permet de rendre le modèle plus robuste en réduisant les poids à chaque étape. Par défaut, ce paramètre est à 0.3, nous avons donc essayé 0.5, 0.7 et 1.

Voici donc ce que nous avons obtenu avec ces changements.

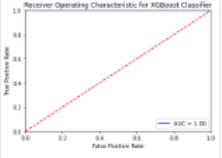
Pour learning_rate = 0.5 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9999	0.9999	0.9999	0.9999	[[179852 10] [3 179133]]	[[99.99, 0.01], [0.0, 100.0]]		501.42

Pour learning_rate = 0.7 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9999	0.9999	0.9999	0.9999	[[179847 15] [4 179132]]	[[99.99, 0.01], [0.0, 100.0]]		479.72

Pour learning_rate = 1 :

Précision	F1 score	Accuracy	Recall	Confusion Matrix	Confusion Matrix (en %)	Plots	Temps (en secs)
0.9999	0.9999	0.9999	0.9999	[[179848 14] [5 179131]]	[[99.99, 0.01], [0.0, 100.0]]		415.10

Comme précédemment, nos tests n'ont rien changés, nous avons donc décidé de laisser les paramètres par défaut du XGBoost.

3. Implémentation du modèle

Nous avons donc instancié un XGBoost Classifier avec ses hyperparamètres par défaut.

```
XGB = xgb.XGBClassifier()
```

Nous l'avons entraîné sur les données d'entraînement :

```
XGB = XGB.fit(X_train,y_train)
```

Les données d'entraînement représentent 80% du dataset que nous avons au préalable équilibré avec l'algorithme SMOTE comme suit :

```
oversample = SMOTE()
X_smote, y_smote = oversample.fit_sample(X, y)

X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)
```

Comme pour les deux autres modèles, nous avons également créé une fonction qui permet de retourner la prédiction de ce modèle sur de nouvelles données et qu'on utilisera à la fin dans l'interface graphique (quand l'utilisateur insère donc de nouvelles données).

```
def XGB_Prediction(df,XGB):
    X_new_XGB = df
    # Predict
    new_prediction_XGB = XGB.predict(X_new_XGB)
    print("New prediction XGB: {}".format(new_prediction_XGB))
    return new_prediction_XGB
```

Cette fonction sera appelée avec le modèle pré-entraîné et sauvegardé grâce à la librairie pickle.

```
# XGB XGBoost Classifier
XGB_DoH = ml.XGB(df_total_csv_normalisee_DoH, colonne)
filename_XGB_DoH = 'DoH/finalized_model_XGB_DoH.sav'
pickle.dump(XGB_DoH, open(filename_XGB_DoH, 'wb'))
```

VI. Deep Learning

A. Théorie

Le Deep Learning est un type de Machine Learning basé sur des réseaux neuronaux artificiels dans lequel de multiples couches de traitement sont utilisées pour extraire des caractéristiques de niveau progressivement plus élevé des données. De nouvelles notions sont présentes dans les réseaux neuronaux : Les fonctions d'activation et la retro-propagation. La fonction d'activation est une fonction mathématique qui permet de traiter l'information qui arrive à un neurone artificiel en Machine Learning, comme le fait ceux du cerveau avec les signaux électriques qu'ils reçoivent. La retro-propagation permet à un algorithme de Deep Learning d'apprendre de ses erreurs en réduisant au maximum la fonction coût à chaque époque parcourue par l'algorithme.

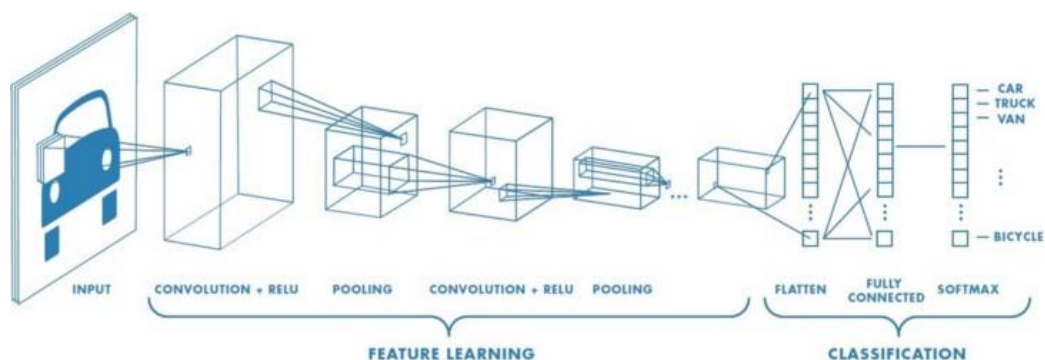
B. Choix des modèles

1. Les réseaux de neurones convolutifs

Les données d'entrée d'un réseau de neurone convolutif sont des images. Dans le cadre de notre projet nous devons donc transformer nos données d'entrée (qui sont des informations réseau) en images sous la forme de matrices de pixels.

Les réseaux de neurones convolutifs disposent de deux parties :

- Partie convolutive : Son objectif est d'extraire des caractéristiques propres à chaque image en réalisant des opérations de réduction sur la taille des images. L'image fournie en entrée passe à travers une succession de filtres qui créent des images de taille réduite.
- Partie classification : L'image réduite par les opérations de convolution passent ensuite dans des couches entièrement connectées MLP (Multi Layers Perceptron). Le rôle de cette partie est de classer l'image.

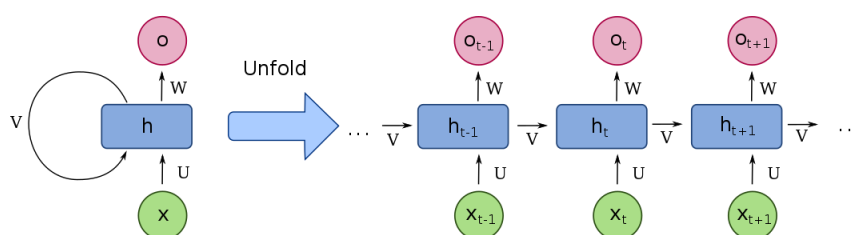


Lors de ce projet, nous nous intéressons au CNN 1D et 2D. Cependant, les CNN partagent les mêmes caractéristiques et suivent la même approche, qu'il s'agisse de 1D, 2D ou 3D. La principale différence est la dimensionnalité des données d'entrée et la façon dont le filtre agit sur les données.

Pour finir, Les CNN ont de nombreuses applications dans la reconnaissance d'images, de vidéos ou le traitement du langage naturel.

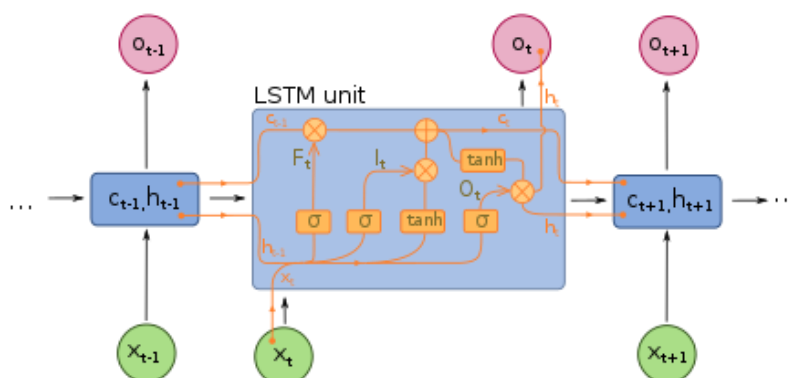
2. Les réseaux de neurones récurrents

Un réseau neuronal récurrent (RNN) est un type de réseau neuronal artificiel. À l'inverse des réseaux de neurones convolutifs (CNN), les réseaux de neurones récurrents utilisent des données d'entraînement pour apprendre : Ils disposent d'une notion de « mémoire ». En effet, ces réseaux agissent sur la prédiction d'une information en gardant en mémoire des informations d'entrée antérieures. Pour résumer, la sortie des réseaux de neurones récurrents dépend des éléments antérieurs passés dans ce réseau.



Une autre caractéristique des réseaux récurrents est qu'ils partagent des paramètres à travers chaque couche du réseau, comme les poids. Cela dit, ces poids seront toujours ajustés par le processus de rétropropagation pour faciliter l'apprentissage par renforcement.

Dans le cadre de ce projet, nous utiliserons le Long Short-Term Memory (LSTM) qui est un type d'architecture de réseau neuronal récurrent artificiel. Celui-ci comporte donc des neurones de rétroaction qui permettent de définir la « mémoire » du modèle. L'intérieur d'un neurone rétroactif est composé de plusieurs fonctions d'activation qui permettent de choisir quelles informations sont à garder ou à jeter en fonction du savoir accumulé préalablement.



Pour finir, les RNN sont couramment utilisés pour des problèmes ordinaux ou temporels, tels que la traduction de la langue, le traitement du langage naturel (NLP), la reconnaissance vocale et le sous-titrage d'images.

C. Les hyperparamètres

Par la suite, nous avons étudié l'optimisation de certains hyperparamètres de nos modèles :

- Nombre de couches de convolution : Cet hyperparamètre définit le nombre de couches de convolution à instancier dans un modèle. Plus le nombre de couches de convolution est importante, plus notre modèle sera complexe.
- Nombre de couches de LSTM : Cet hyperparamètre définit le nombre de couches RNN à instancier dans un modèle. Plus le nombre de couches de convolution est importante, plus notre modèle sera complexe.
- Nombre de filtres : Cet hyperparamètre définit le nombre de filtres à appliquer à chaque couche de convolution. Plus il y'a de filtres, plus le nombre de convolutions est importante.
- Le nombre d'unités : Le nombre d'unités définit le nombre de neurones rétroactifs dans chaque couche de LSTM.
- Dropout : La méthode du dropout consiste à « désactiver » des sorties de neurones aléatoirement pendant la phase d'apprentissage. Cela permet de contrôler le sur-apprentissage d'un modèle.
- Batch size : Le Batch size est un terme utilisé dans l'apprentissage automatique et fait référence au nombre d'exemples d'apprentissage utilisés dans une itération. Il y'aura donc plus d'itérations si le nombre d'exemples est petit.
- Epochs : Le nombre d'époques est un hyperparamètre qui définit le nombre de fois que l'algorithme d'apprentissage travaillera sur l'ensemble des données d'apprentissage. La notion de rétropropagation intervient entre chaque époque.

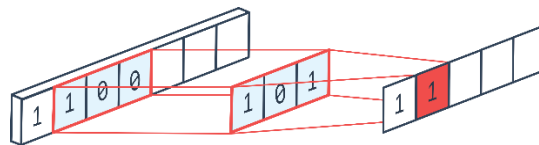
D. CNN 1D

1. Explication du modèle

Comme dit précédemment, La principale différence entre le CNN 1D et le CNN 2D est la dimensionnalité des données d'entrée et la façon dont le filtre agit sur les données.

Pour un CNN 1D, la dimension d'une image en entrée est de 2 dimensions. La première correspond à nos features contenues dans nos données d'entraînement (33), et la deuxième correspond à la dimension temporelle des données. Concrètement, cela indique si notre image est en niveaux de gris ou en couleurs. Dans notre cas, nous considérerons que les images créées à l'aide de nos données sont en niveaux de gris (donc de dimension 1). Nous avons donc en entrée une image qui est une matrice de dimensions 33x1 et nous devons maintenant la faire passer dans une série de convolutions.

Les opérations de convolutions appliquent des filtres sur l'image pour en réduire sa dimension. Comme notre image est de taille 33x1, nous instancions un filtre de taille 3x1.



2. Implémentation du modèle

Voici le modèle que nous avons obtenu après optimisation des hyperparamètres :

```
def model_cnn_1D(df, colonne, nb_layers, first_layer_nb_filters, layer_nb_filters, dropout_alpha, filter_size, nb_epochs, batch_size):  
    X = df.drop([colonne], axis = 1)  
    y = df[colonne]  
  
    oversample = SMOTE()  
    X_smote, y_smote = oversample.fit_sample(X, y)  
  
    X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=1)  
  
    X_train = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1)  
    X_test = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)  
  
    t_debut = time.time()  
  
    model = Sequential()  
  
    model.add(Conv1D(filters=first_layer_nb_filters, kernel_size=filter_size, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))  
    if nb_layers != 0:  
        for i in range(0, nb_layers):  
            model.add(Conv1D(filters=layer_nb_filters, kernel_size=filter_size, activation='relu'))  
  
    model.add(Dropout(dropout_alpha))  
    model.add(MaxPooling1D(pool_size=2))  
    model.add(Flatten())  
    model.add(Dense(32, activation='relu'))  
    model.add(Dense(1, activation='sigmoid'))  
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Les hyperparamètres instanciés dans ce modèle sont :

- Nombre de couches de convolution : 3
- Nombre de filtres : 32 + 16 + 16 (pour les 3 couches de convolution)
- Dropout : 0.5
- Batch size : 64
- Epochs : 8

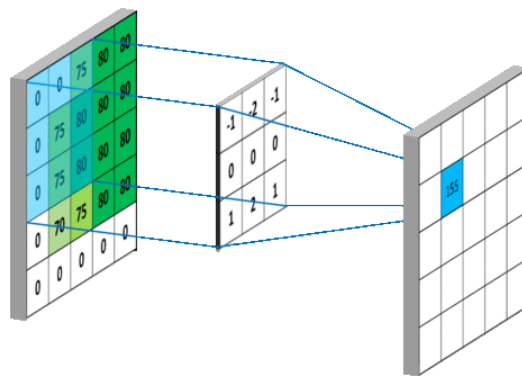
E. CNN 2D

1. Explication du modèle

Pour un CNN 2D, la dimension d'une image en entrée est de 3 dimensions. Celles-ci sont la largeur, la hauteur et la dimension temporelle des données (toujours fixée à 1 car l'image est en niveau de gris).

Pour la largeur et la hauteur, nous répartissons gardons les 33 features pour la largeur et définissons 1 pixel pour la hauteur.

Pour les filtres, un CNN 2D requiert un filtre de dimension 2. Nous avons alors défini un filtre 3x3 en largeur et en hauteur. Cependant, puisque la taille de notre image d'entrée est de 33x1, nous utilisons la paramètre « padding = 'same' » dans les convolutions 2D, ce qui permet d'effectuer un rembourrage sur notre image. Cela permet donc au filtre de s'adapter à la taille de notre image d'entrée.



2. Implémentation du modèle

Voici le modèle que nous avons obtenu après optimisation des hyperparamètres :

```
def model_cnn_2D(df, colonne, nb_layers, first_layer_nb_filters, layer_nb_filters, nb_epochs, batch_size):

    X = df.drop([colonne], axis = 1)
    y = df[colonne]

    oversample = SMOTE()
    X_smote, y_smote = oversample.fit_sample(X, y)

    scaler = MinMaxScaler(feature_range=(0, 255))

    scaler = scaler.fit(X_smote)
    X_scaled = scaler.transform(X_smote)

    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_smote, test_size=0.2, random_state=1)
    X_train = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1, 1)

    X_test = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1, 1)

    t_debut = time.time()

    model = Sequential()
    model.add(Conv2D(first_layer_nb_filters, (3, 3), activation='relu', input_shape=(33, 1, 1), padding='same'))
    model.add(MaxPooling2D((2, 2), padding='same'))

    if nb_layers != 0:
        for i in range(0, nb_layers):
            model.add(Conv2D(layer_nb_filters, (3, 3), activation='relu', padding='same'))
            model.add(MaxPooling2D((2, 2), padding='same'))
            print('couche ajoutée')

    model.add(Flatten())
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    model.summary()
```

Les hyperparamètres que nous avons instancié dans ce modèle sont :

- Nombre de couches de convolution : 2
- Nombre de filtres : 32 + 64 (pour les 2 couches de convolution)
- Batch size : 256
- Epochs : 8

F. LSTM

1. Explication du modèle

Les dimensions à donner en entrée d'un LSTM sont de 3. La première est le nombre d'échantillons. Le deuxième est le paramètre timestep. Le paramètre timestep désigne le nombre d'occurrences précédentes que le modèle examine pour effectuer la prédiction. Nous l'avons instancié à 33. La troisième dimension est le nombre de features (une feature est une observation à un pas de temps). Nous l'instancions à 1.

2. Implémentation du modèle

Voici le modèle que nous avons obtenu après optimisation des hyperparamètres :

```
def model_lstm(df, colonne, nb_layers, first_layer_nb_neurons, other_layer_nb_neurons, dropout_alpha, nb_epochs, batch_size):  
    X = df.drop([colonne], axis = 1)  
    y = df[colonne]  
  
    oversample = SMOTE()  
    X_smote, y_smote = oversample.fit_sample(X, y)  
  
    X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=1)  
  
    X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))  
    X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)  
  
    t_debut = time.time()  
  
    model_lstm = Sequential()  
  
    model_lstm.add(LSTM(first_layer_nb_neurons, return_sequences = True, input_shape=(33,1)))  
    model_lstm.add(Dropout(dropout_alpha))  
  
    if nb_layers != 0:  
        for i in range(0,nb_layers):  
            model_lstm.add(LSTM(other_layer_nb_neurons, return_sequences = True))  
            model_lstm.add(Dropout(dropout_alpha))  
            print('couche ajoutée')  
  
    model_lstm.add(Flatten())  
    model_lstm.add(Dense(32, activation = 'relu'))  
  
    model_lstm.add(Dense(1, activation='sigmoid'))  
  
    #sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)  
    model_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])  
    model_lstm.summary()
```

Les hyperparamètres que nous avons instanciés dans notre LSTM sont :

- Nombre de couches de LSTM : 2
- Nombre d'unités : 50
- Dropout : 0.5
- Batch size : 256
- Epochs : 5

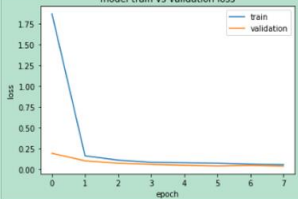
G. Résultats et comparaisons

Pour évaluer nos modèles, nous nous sommes basés sur certaines métriques essentielles pour obtenir un bon modèle :

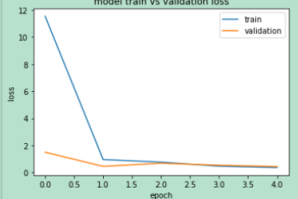
- Précision : La précision est définie comme le rapport entre les vrais positifs et le nombre total de points de données prédits comme positifs par un modèle.
- Accuracy : L'accuracy est définie comme le nombre de vrais positifs et de vrais négatifs divisé par le nombre total d'occurrences.
- F1-Score : Le score F1 est défini comme la moyenne harmonique pondérée de la précision et du rappel du test.
- Confusion Matrix : Une matrice de confusion est une technique permettant de résumer les performances d'un algorithme de classification. Le calcul d'une matrice de confusion peut vous donner une meilleure idée de ce qu'un modèle et des types d'erreurs qu'il commet.
- Diagramme Loss / Epochs : Ce diagramme permet de visualiser la tendance de l'erreur d'un modèle au fil des époques parcourues.
- Le temps d'exécution : Le temps d'exécution de l'entraînement d'un modèle est important car le Deep Learning utilise beaucoup de ressources de calculs.

Voici les métriques des 3 modèles de Deep Learning optimisés :

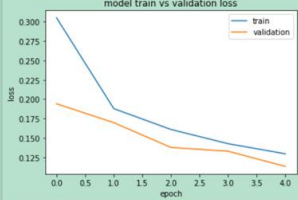
CNN 1D

Precision	F1 score	Accuracy	Confusion Matrix (en %)	Plots (loss)	Temps (en sec)
0.9903	0.9916	0.9916	[[99.03, 0.97], [0.7, 99.3]]		281.32

CNN 2D

Precision	F1 score	Accuracy	Confusion Matrix (en %)	Plots (loss)	Temps (en sec)
0.9609	0.9592	0.9593	[[96.1, 3.9], [4.24, 95.76]]		249.1496

LSTM

Precision	F1 score	Accuracy	Confusion Matrix (en %)	Plots (loss)	Temps (en sec)
0.9428	0.9595	0.9587	[[94.07, 5.93], [2.32, 97.68]]		754.26

Ces trois modèles ont tous des très bonnes métriques en termes de précision, F1 score et accuracy.

Nous pouvons voir d'après la matrice de confusion que le CNN 1D est celui qui se trompe le moins souvent. Les diagrammes Plot / Loss nous indiquent que les modèles CNN 1D et CNN 2D sont des bons modèles qui ont appris correctement : la tendance d'erreur entre les données de train et de validation sont similaires. Cependant, le LSTM est un modèle sous appris car la tendance des deux courbes d'erreur est décroissante et peuvent apprendre davantage.

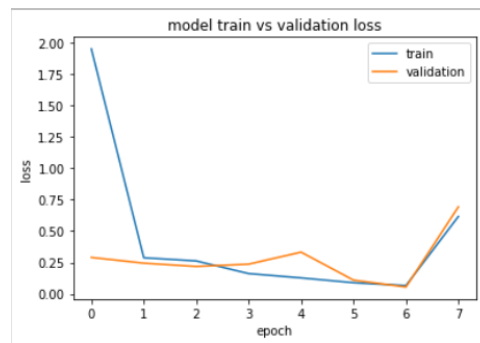
Les temps d'exécution du modèle CNN 1D et CNN 2D sont de moins de 5 minutes, ce qui est convenable. En revanche, le modèle LSTM met plus de 12 minutes pour s'exécuter. D'autant plus que celui-ci effectue un sous apprentissage.

D'après ces points, nous décidons par la suite de garder le CNN 1D et le CNN 2D pour établir notre ferme de modèle de Deep Learning pour la détection d'intrusions dans un réseau.

Remarques

Voici quelques exemples qui justifient le choix de ces modèles entraînés avec les hyperparamètres spécifiques :

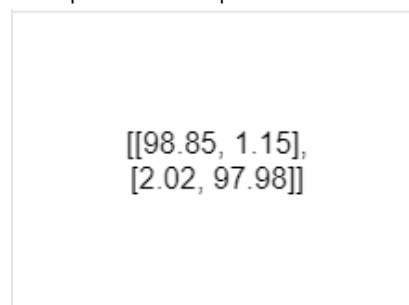
- Sur apprentissage du modèle CNN 2D : Lors de nos essais, nous avons entraîné le modèle CNN 2D avec une troisième couche de convolution instanciée avec une taille de 64 filtres. Nous avons constaté que ce modèle était en sur-apprentissage d'après le diagramme Loss / Epochs suivant :



Les courbes de train et de validation effectuent plus d'erreurs lors du passage de la 6^{ème} à la 7^{ème} époque. Cela se traduit par un sur-apprentissage du modèle.

- Matrice de confusion : L'élément le plus important de la matrice de confusion est le taux de faux positifs. Dans notre cas, celui-ci mesure le nombre de fois qu'une intrusion a été détectée comme un trafic bénin. C'est ce que nous voulons absolument éviter car c'est dans ces cas que l'on expose les utilisateurs aux risques des cyber-attaques. Cet élément a été très important pour la sélection de nos modèles optimisés.

Par exemple, voici la matrice de confusion d'un modèle CNN 1D avec différents hyperparamètres que nous ne spécifierons pas :



Le taux de faux positifs (en bas à gauche) est supérieur à celui des faux négatifs (en haut à droite). Les métriques du modèle sont très bonnes mais avec cet élément précis, nous n'avons pas retenu le modèle.

VII. Interfaces

A. Application en local PySimpleGUI

Pour l'interface graphique, nous avons utilisé la librairie PySimpleGUI, pour que les résultats soient présentés de manière plus interactive à l'utilisateur.

1. Interface de présentation et du choix

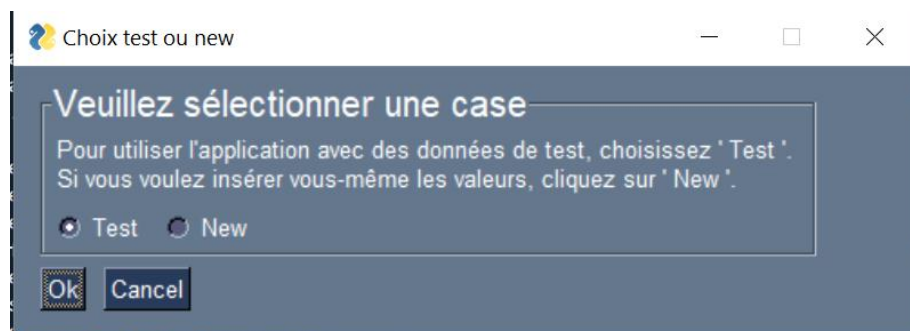
Tout d'abord, nous avons créé une interface pour laisser le choix à l'utilisateur s'il désire tester des données qu'il possède dans un fichier .csv ou s'il préfère insérer à la main ses propres données réseau. Nous avons donc dans cette première interface, deux boutons radio créés avec `sg.Radio()`, dans une frame qui par la suite est intégrée à la fenêtre 'Choix test ou new', comme on peut le voir sur l'image ci-dessous.

```
def beginInterface():
    sg.theme('DarkBlue3')

    frame_layout = [[sg.Text("Pour utiliser l'application avec des données de test, choisissez 'Test'."),
                     [sg.Radio('Test', "RADIO1", default=True),
                      sg.Radio('New', "RADIO1")]]

    layout = [
        [sg.Frame('Veuillez sélectionner une case', frame_layout, font=15)],
        [sg.Button('Ok'), sg.Button('Cancel')]
    ]
    window = sg.Window('Choix test ou new', layout, size=(500,150))
```

Ce code permet donc d'afficher cette interface à l'utilisateur :



La suite de cette fonction `beginInterface()` se présente comme ceci :

```
radio_value=''

while (True):
    event, values = window.read()
    if event == sg.WIN_CLOSED or event == 'Cancel':
        radio_value='New'
        window.close()
        break
    elif event == 'Ok':
        print(values)
        if values[0] == True:
            radio_value = 'Test'
        elif values[1] == True:
            radio_value = 'New'
        window.close()
        break
return radio_value
```

C'est-à-dire que nous récupérons la valeur des boutons radio lorsque l'utilisateur clique sur le bouton 'ok' pour que par la suite savoir quelle interface lui montrer.

2. Interface insertion des données réseaux : choix Test

Si l'utilisateur choisit 'Test', c'est-à-dire nous regardons la valeur du bouton radio, nous choisissons alors au hasard parmi 10 fichiers csv que nous avons localement, un que nous testerons.

```
if radio_value == 'Test':  
    nb = int(np.random.randint(1,11))  
    df = pd.read_csv("Tests_interface/df_test_"+str(nb)+".csv", sep=';')
```

Ensuite nous pouvons donc appeler notre interface d'insertion des données avec cette dataframe en paramètre :

```
# On appelle l'interface principale  
button_value = gui.interface(df)
```

Cette interface apparaîtra, avec les données rentrées automatiquement à partir du fichier csv sélectionné aléatoirement :

Veuillez rentrer vos informations	
SourceIP	192.168.20.2
DestinationIP	1.1.1.1
SourcePort	43722
DestinationPort	443
Duration	120.666681
FlowBytesSent	50093
FlowSentRate	415.1353098
FlowBytesReceived	91716
FlowReceivedRate	760.0772577
PacketLengthVariance	15037.42315
PacketLengthStandardDeviation	122.6271713
PacketLengthMean	128.9172727
PacketLengthMedian	87.0
PacketLengthMode	87
PacketLengthSkewFromMedian	1.025480868
PacketLengthSkewFromMode	0.341826956
PacketLengthCoefficientofVariation	0.951208234
PacketTimeVariance	1190.218534
PacketTimeStandardDeviation	34.49954397
PacketTimeMean	64.25283658
PacketTimeMedian	67.9760645
PacketTimeMode	0.0
PacketTimeSkewFromMedian	-0.32376322
PacketTimeSkewFromMode	1.862425678
PacketTimeCoefficientofVariation	0.536934177
ResponseTimeVariance	0.375993471
ResponseTimeStandardDeviation	0.613183068
ResponseTimeMean	0.156775561
ResponseTimeMedian	0.001646999
ResponseTimeMode	1.3e-5
ResponseTimeSkewFromMedian	0.758966958
ResponseTimeSkewFromMode	0.255653768
ResponseTimeCoefficientofVariation	3.911215877

Ok Cancel

S'il le souhaite, l'utilisateur peut modifier ces données.

Nous avons affiché cette interface avec le code suivant :

Tout d'abord, nous affichons dans une frame les noms des features et les champs de saisie remplis parce que la dataframe n'est pas vide, séparés en deux colonnes.

```
if not df.empty:  
    for i in range(0, len(liste_colonne)):  
        frame_layout.append([sg.Text(liste_colonne[i], size=(30, 1)),  
                             sg.InputText(default_text = df[list_colonne[i]].values[0], size=(10, 1))])
```

Puis nous ajoutons deux boutons : 'Ok' et 'Cancel' et insérons le tout à la fenêtre 'Informations pour prédictions'.

```
layout = [
    [sg.Frame('Veuillez rentrer vos informations', frame_layout, font=15)],
    [sg.Button('Ok'), sg.Button('Cancel')]
]

window = sg.Window('Informations pour prediction', layout, size=(750,520))
```

Pour finir, une fois que l'utilisateur appuie sur le bouton 'Ok', nous enregistrons les données insérées dans une nouvelle dataframe pour pouvoir l'utiliser par la suite.

Si l'utilisateur appuie sur 'Cancel' la fenêtre se ferme et le code s'arrête. S'il appuie sur 'Ok', nous enregistrons les données entrées par l'utilisateur dans une nouvelle dataframe, pour par la suite effectuer des prédictions dessus.

```
button_value=''
while (True):
    event, values = window.read()
    if event == sg.WIN_CLOSED or event == 'Cancel':
        button_value='Cancel'
        window.close()
        break

    elif event == 'Ok':
        X_new = []
        for j in range(0, len(liste_colonne)):
            X_new.append(values[j])
        features=[]
        for i in range(0,len(liste_colonne)):
            features.append(liste_colonne[i])

        df_new = pd.DataFrame([X_new], columns=features)
        df_new.to_csv('df_new.csv', sep=';', index = False)
        button_value='Ok'
        window.close()
        break
```

3. Interface insertion des données réseaux : choix New

Si l'utilisateur choisit 'New', nous appelons comme précédemment la fonction 'interface'.

```
# On appelle l'interface principale  
button_value = gui.interface(df)
```

Mais cette fois-ci le champ 'df' sera vide, nous afficherons donc aussi le nom des features mais cette fois-ci avec des champs de saisie vides.

```
else:  
    for i in range(0, len(liste_colonne)):  
        frame_layout.append([sg.Text(liste_colonne[i], size=(30, 1)),  
                             sg.InputText(default_text = 0, size=(10, 1))])
```

Nous obtenons donc cette interface :

Informations pour prediction

Veuillez rentrer vos informations

SourceIP	0	PacketTimeVariance	0
DestinationIP	0	PacketTimeStandardDeviation	0
SourcePort	0	PacketTimeMean	0
DestinationPort	0	PacketTimeMedian	0
Duration	0	PacketTimeMode	0
FlowBytesSent	0	PacketTimeSkewFromMedian	0
FlowSentRate	0	PacketTimeSkewFromMode	0
FlowBytesReceived	0	PacketTimeCoefficientofVariation	0
FlowReceivedRate	0	ResponseTimeTimeVariance	0
PacketLengthVariance	0	ResponseTimeTimeStandardDeviation	0
PacketLengthStandardDeviation	0	ResponseTimeTimeMean	0
PacketLengthMean	0	ResponseTimeTimeMedian	0
PacketLengthMedian	0	ResponseTimeTimeMode	0
PacketLengthMode	0	ResponseTimeTimeSkewFromMedian	0
PacketLengthSkewFromMedian	0	ResponseTimeTimeSkewFromMode	0
PacketLengthSkewFromMode	0	ResponseTimeTimeCoefficientofVariation	0
PacketLengthCoefficientofVariation	0		

Ok Cancel

Puis nous avons ajouté également les boutons 'Ok' et 'Cancel'. Et nous enregistrons les données insérées dans une nouvelle dataframe.

4. Interface résultat DoH

Lorsque l'utilisateur clique sur 'Ok', il arrive sur l'interface des résultats des prédictions pour savoir s'il s'agit d'un DoH ou non DoH.

Avant toute chose, nous traitons les données insérées. C'est-à-dire qu'on supprime les colonnes inutiles, convertit les IP, et les normalise.

```
for colonne in df_test.columns:
    if colonne == '' or colonne == 'Timestamp' or colonne == 'TimeStamp' or colonne == 'index' or
        del df_test[colonne]
for colonne in df_test.columns:
    if type(df_test[colonne][0]) == str and colonne != 'SourceIP' and colonne != 'DestinationIP':
        df_test[colonne][0] = 0

if type(df_test['SourceIP']) != float:
    df_test = pc.IP2Int(df_test, 'SourceIP')
if type(df_test['DestinationIP']) != float:
    df_test = pc.IP2Int(df_test, 'DestinationIP')
df_test = pc.nettoyage(df_test)
df_test=nor.NormalizeNewValues(original_dataset, df_test)

for colonne in df_test.columns:
    if(colonne != 'SourceIP' and colonne != 'DestinationIP'):
        df_test[colonne] = float(df_test[colonne])
```

Pour cette nouvelle dataframe, on appelle donc la fonction de prédiction de chaque modèle, expliquée précédemment dans les modèles de Machine Learning.

```
# Stocke predictions
pred_DTC_DoH = ml.DTC_Prediction(df_test, loaded_model_DTC_DoH)
pred_XGB_DoH = ml.XGB_Prediction(df_test, loaded_model_XGB_DoH)
pred_RFC_DoH = ml.RFC_Prediction(df_test, loaded_model_RFC_DoH)
```

Ces dernières sont stockées pour être appelées et affichées dans une pop-up que nous définissons comme suit :

```
if(compteur >= 2):
    sg.popup('Resultat DoH', 'Le resultat pour Le DTC :{}'.format(resultat_DTC_DoH),
        'Le resultat pour Le RFC :{}'.format(resultat_RFC_DoH),
        'Le resultat pour Le XGB :{}'.format(resultat_XGB_DoH),

        'Le resultat pour Le GNB :{}'.format(resultat_GNB_DoH),
        'Le resultat pour Le KNN :{}'.format(resultat_KNN_DoH),
        'Le resultat pour Le Perceptron :{}'.format(resultat_Per_DoH),
        '\n{} modeles prédisent un DoH, Les données insérées représentent donc un DoH'.format(compteur))
    result_intrusion(pred_Conv1D_Model_Intrusion,pred_Conv2D_Model_Intrusion)
else:
    sg.popup('Resultat DoH', 'Le resultat pour Le DTC :{}'.format(resultat_DTC_DoH),
        'Le resultat pour Le RFC :{}'.format(resultat_RFC_DoH),
        'Le resultat pour Le XGB :{}'.format(resultat_XGB_DoH),

        'Le resultat pour Le GNB :{}'.format(resultat_GNB_DoH),
        'Le resultat pour Le KNN :{}'.format(resultat_KNN_DoH),
        'Le resultat pour Le Perceptron :{}'.format(resultat_Per_DoH),
        '\n{} modèle prédit un DoH, Les données insérées représentent donc un non DoH'.format(compteur))
```

On a instancié un compteur qui s'incrémente si les modèles ont prédit que les données représentaient un DoH, voici un exemple pour un modèle :

```

compteur=0

if(pred_DTC_DoH == 1.0):
    resultat_DTC_DoH = 'DoH'
    compteur+=1
else:
    resultat_DTC_DoH = 'nonDoH'

```

Si ce compteur est supérieur à 2, alors on en déduit que les données représentent un DoH, et on peut continuer les prédictions pour savoir s'il s'agit d'une Intrusion. Sinon le programme s'arrête.

Voici l'interface que l'utilisateur obtient pour connaître s'il s'agit d'un DoH :



5. Interface résultat Intrusion

Pour finir, s'il s'agit d'un non DoH comme sur la première image au-dessus alors le programme s'arrête. S'il s'agit d'un DoH, comme sur la deuxième image alors on lance les prédictions des modèles de Deep Learning sur les données comme suit :

```
pred_Conv1D_Model_Intrusion = dl.Conv1D_Prediction(df_test, loaded_model_Conv1D_Model_Intrusion)
pred_Conv2D_Model_Intrusion = dl.Conv2D_Prediction(df_test, loaded_model_Conv2D_Model_Intrusion)
```

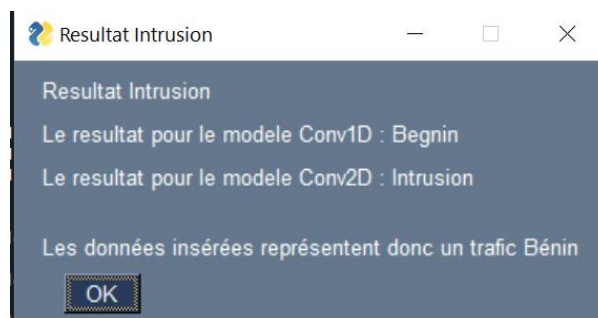
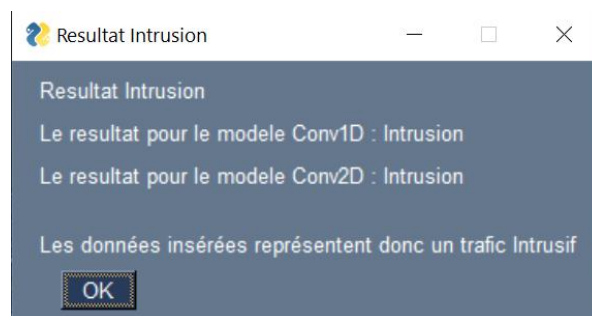
Et on affiche également une pop-up instanciée avec le code suivant :

```
compteur = 0
if(pred_Conv1D_Model_Intrusion == 1.0):
    resultat_Conv1D_Model_Intrusion = 'Intrusion'
    compteur+=1
else:
    resultat_Conv1D_Model_Intrusion = 'Benin'

if(pred_Conv2D_Model_Intrusion == 1.0):
    resultat_Conv2D_Model_Intrusion = 'Intrusion'
    compteur+=1
else:
    resultat_Conv2D_Model_Intrusion = 'Benin'

if(compteur == 2):
    sg.popup('Resultat Intrusion',
            'Le resultat pour le modele Conv1D : {}'.format(resultat_Conv1D_Model_Intrusion),
            'Le resultat pour le modele Conv2D : {}'.format(resultat_Conv2D_Model_Intrusion),
            '\nLes données insérées représentent donc un trafic Intrusif')
else:
    sg.popup('Resultat Intrusion',
            'Le resultat pour le modele Conv1D : {}'.format(resultat_Conv1D_Model_Intrusion),
            'Le resultat pour le modele Conv2D : {}'.format(resultat_Conv2D_Model_Intrusion),
            '\nLes données insérées représentent donc un trafic Bénin')
```

Nous spécifions donc si le trafic est Bénin ou Intrusif, l'utilisateur obtient donc cette interface :



B. Application Web Flask

Pour cette interface, le processus est le même pour déterminer s'il s'agit d'un DoH ou non puis d'une Intrusion ou non, nous allons seulement voir le côté technique. Nous avons donc créé un environnement pour notre projet.

1. Mise en place de la base de données

Nous avons utilisé une base de données SQLite pour stocker nos données, car le module `sqlite3` est disponible dans la bibliothèque standard de Python.

Nous commençons donc par créer notre schéma en `.sql` qui contient les commandes SQL pour créer notre tableau avec les colonnes que nous désirons.

```
DROP TABLE IF EXISTS posts;

CREATE TABLE posts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    title TEXT NOT NULL,
    ok VARCHAR(255),
    SourceIP FLOAT(64),
    DestinationIP FLOAT(64),
    SourcePort INT(64),
    DestinationPort INT(64),
    Duration FLOAT(64),
```

Nous avons donc un `id` qui représente la clé primaire, à laquelle la base de données attribuera une valeur unique et qui s'incrémentera à chaque nouvelle insertion.

Nous spécifions aussi l'heure à laquelle nous insérons une nouvelle donnée, et un titre pour que ce soit plus facile de s'y retrouver.

Puis nous avons spécifié toutes les features de nos datasets ainsi que les prédictions à venir.

La connexion à la base de données se fait à partir de cette fonction :

```
def get_db_connection():
    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row
    return conn
```

2. Page d'accueil

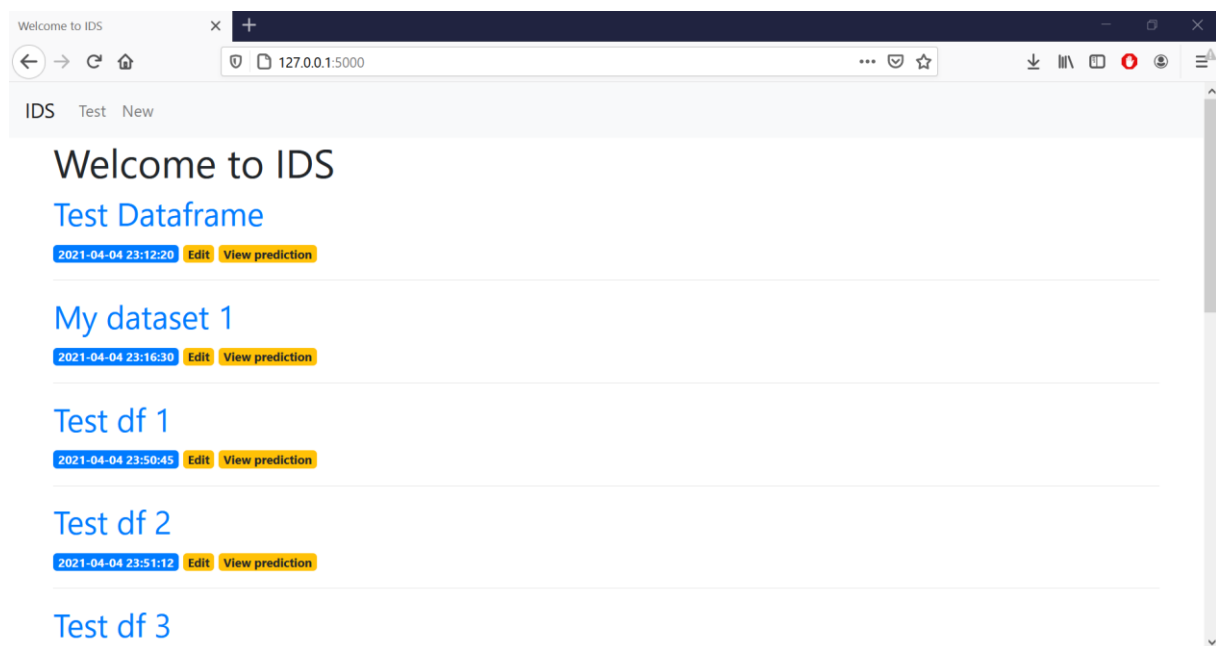
La page d'accueil apparaîtra, et nous afficherons tous les tests, donc les données insérées, grâce à la fonction suivante :

```
@app.route('/')
def index():
    conn = get_db_connection()
    posts = conn.execute('SELECT * FROM posts').fetchall()
    conn.close()
    return render_template('index.html', posts=posts)
```

Nous faisons donc une requête SQL pour afficher tout ce qu'il y a dans la base de données.

Nous avons créé un fichier HTML et utilisé bootstrap pour le rendu (voir le code).

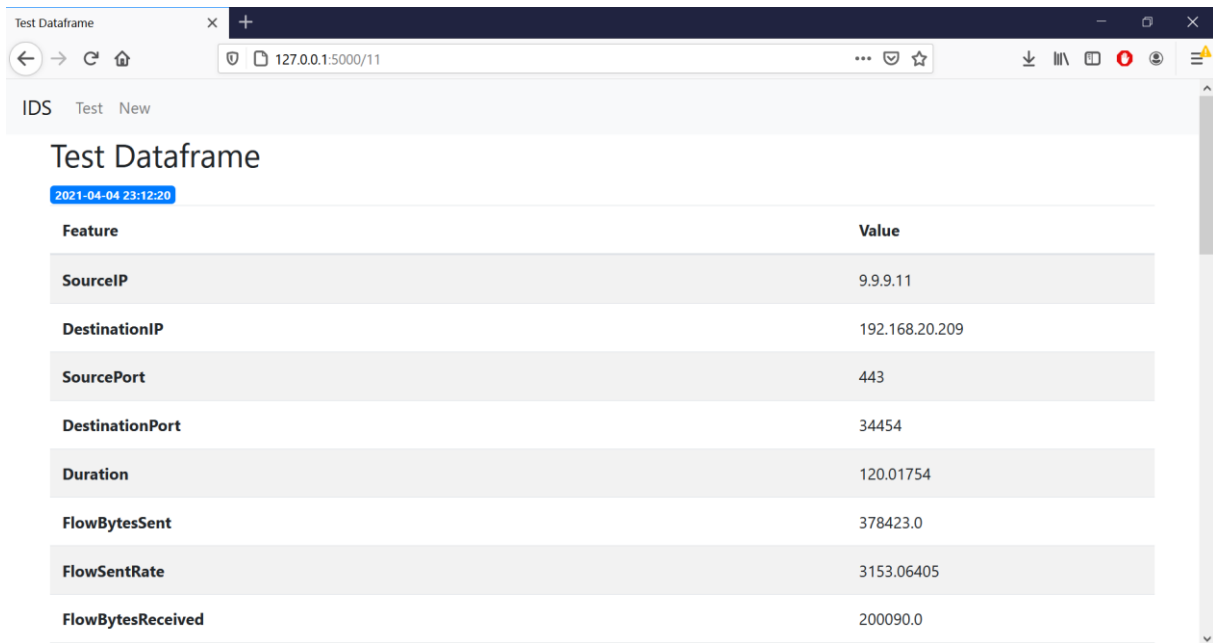
Le rendu de cette page est comme suit :



Nous pouvons donc voir l'historique des tests effectués. Nous avons également un onglet 'Test' et un autre 'New' qui suit le même principe que l'interface précédente, et que nous allons détailler par la suite.

3. Affichage des données insérées

A partir de la page d'accueil, nous pouvons cliquer sur le titre du test par exemple 'Test Dataframe', qui nous conduira à la page suivante :



Feature	Value
SourceIP	9.9.9.11
DestinationIP	192.168.20.209
SourcePort	443
DestinationPort	34454
Duration	120.01754
FlowBytesSent	378423.0
FlowSentRate	3153.06405
FlowBytesReceived	200090.0

Nous pouvons donc voir les données que nous avons insérées.

Ceci s'effectue grâce au code suivant :

```
def get_post(post_id):
    conn = get_db_connection()
    post = conn.execute('SELECT * FROM posts WHERE id = ?',
                        (post_id,)).fetchone()
    conn.close()
    if post is None:
        abort(404)
    return post
```

Tout d'abord nous avons cette fonction qui permet de déterminer quel test doit être renvoyé, on se connecte donc à la base de donnée avec la fonction `get_db_connection()` et on exécute la requête SQL pour obtenir les données associées à la valeur `post_id`. Si la requête stockée dans `post` est vide, nous répondons avec le code d'erreur 404 et la fonction s'arrête. Et si la requête a abouti, et que la variable `post` n'est pas vide on renvoie la valeur de cette dernière.

Pour l'utiliser dans la fonction suivante qui va nous permettre d'afficher le résultat.

```
@app.route('/<int:post_id>')
def post(post_id):
    post = get_post(post_id)
    return render_template('post.html', post=post)
```

Nous obtenons grâce au code HTML l'interface présentée au début.

4. Insertion de nouvelles données (NEW)

Pour insérer de nouvelles données et connaître le résultat des prédictions par la suite, nous avons créé un formulaire en HTML où l'utilisateur doit spécifier une valeur pour chaque feature.

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/new'. The page title is 'Create your own dataset'. The form contains the following fields:

- Title
- SourceIP
- DestinationIP
- SourcePort
- DestinationPort
- Duration
- ResponseTimeTimeMean
- ResponseTimeTimeMedian
- ResponseTimeTimeMode
- ResponseTimeTimeSkewFromMedian
- ResponseTimeTimeSkewFromMode
- ResponseTimeTimeCoefficientofVariation

A blue 'Submit' button is located at the bottom of the form.

Une fois que l'utilisateur a bien rempli les champs, et qu'il clique sur 'Submit', on insère ces données dans la base de données :

```
conn = get_db_connection()
conn.execute('INSERT INTO posts (title,ok, SourceIP, DestinationIP, SourcePort,
                                (title, 'ok', SourceIP, DestinationIP, SourcePort, DestinationPort,
                                Duration) VALUES (?, ?, ?, ?, ?, ?)')

id = conn.execute('SELECT id FROM posts WHERE title=? AND SourceIP=? AND Destin
                  (title, SourceIP, DestinationIP, SourcePort, DestinationPort, Durat
```

On récupère également l'id créé à son insertion pour pouvoir enregistrer un fichier CSV dans nos dossiers avec cet id :

```
df_test = pd.DataFrame([SourceIP, DestinationIP, SourcePort, DestinationPort,
df_test.to_csv('df/df_test_'+str(id)+'.csv', sep=';')
```

Nous faisons également sur cette dataset le même traitement effectué sur l'autre interface, c'est-à-dire que nous supprimons les colonnes inutiles, convertit les IP en float, nettoie et normalise les données afin d'effectuer les prédictions dessus. De plus, nous mettons à jour dans la base de données ces données traitées.

```
for colonne in df_test.columns:
    if colonne == '' or colonne == 'Timestamp' or colonne == 'TimeStamp' or col
        del df_test[colonne]
for colonne in df_test.columns:
    if type(df_test[colonne][0]) == str and colonne != 'SourceIP' and colonne !=
        df_test[colonne][0] = 0

conn = get_db_connection()
conn.execute('UPDATE posts SET SourceIP=?, DestinationIP=? WHERE id=?',
              (df_test['SourceIP'][0], df_test['DestinationIP'][0], id))

if type(df_test['SourceIP']) != float:
    df_test = pc.IP2Int(df_test, 'SourceIP')
if type(df_test['DestinationIP']) != float:
    df_test = pc.IP2Int(df_test, 'DestinationIP')
df_test = pc.nettoyage(df_test)
df_test = nor.NormalizeNewValues(original_dataset, df_test)

for colonne in df_test.columns:
    if (colonne != 'SourceIP' and colonne != 'DestinationIP'):
        df_test[colonne] = float(df_test[colonne])
        conn.execute('UPDATE posts SET '+colonne+'=? WHERE id=?',
                      (df_test[colonne][0], id))
```

Ensuite, toujours dans la même fonction nous stockons les prédictions pour pouvoir les insérer dans la base de données.

```
pred_DTC_DoH = ml.DTC_Prediction(df_test, loaded_model_DTC_DoH)
pred_XGB_DoH = ml.XGB_Prediction(df_test, loaded_model_XGB_DoH)
pred_RFC_DoH = ml.RFC_Prediction(df_test, loaded_model_RFC_DoH)

#df_test_norm = nor.NormalizeDataset(df_test)
#pred_GNB_DoH = ml_bis.GNB_Prediction(df_test_norm, loaded_model_GNB_DoH)
pred_GNB_DoH = 0
#pred_KNN_DoH = ml_bis.KNN_Prediction(df_test_norm, loaded_model_KNN_DoH)
pred_KNN_DoH = 0

pred_Conv1D_Model_Intrusion = dl.Conv1D_Prediction(df_test, loaded_model_Conv1D)
pred_Conv2D_Model_Intrusion = dl.Conv2D_Prediction(df_test, loaded_model_Conv2D)

#conn = get_db_connection()
conn.execute('UPDATE posts SET pred_DTC_DoH = ?, pred_RFC_DoH = ?, pred_XGB_DoH
              ' WHERE id = ?',
              (int(pred_DTC_DoH), int(pred_RFC_DoH), int(pred_XGB_DoH), int(pred_KN
```

Pour finir nous avons instancié un compteur pour déterminer s'il s'agit d'un DoH puis d'une Intrusion :

```
compteur_DoH=0
compteur_Intrusion=0

if(pred_DTC_DoH == 1.0):
    compteur_DoH+=1

if(pred_RFC_DoH == 1.0):
    compteur_DoH+=1

if(pred_XGB_DoH == 1.0):
    compteur_DoH+=1

if(pred_GNB_DoH == 1.0):
    compteur_DoH+=1

if(pred_KNN_DoH == 1.0):
    compteur_DoH+=1

if(pred_Conv1D_Model_Intrusion == 1.0):
    compteur_Intrusion+=1

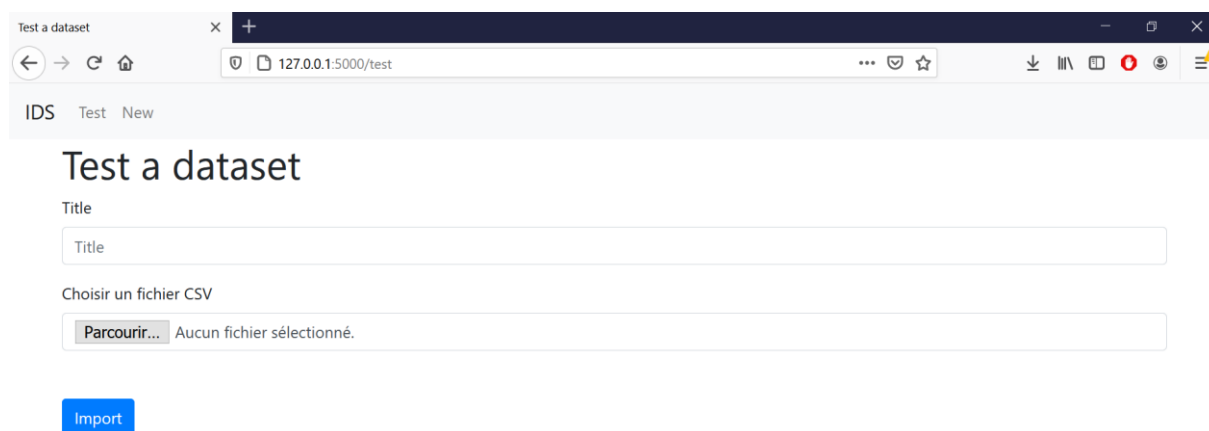
if(pred_Conv2D_Model_Intrusion == 1.0):
    compteur_Intrusion+=1
```

Nous ajoutons ce compteur à la base de données.

Lorsque toutes ces étapes sont finies, l'utilisateur est redirigé vers une page où il peut voir le résultat des prédictions (nous verrons cette interface par la suite).

5. Insertion de nouvelles données (TEST)

Sur la page d'accueil, l'utilisateur a aussi le choix d'insérer des données à partir d'un onglet 'Test', ce dernier sera redirigé vers cette page :

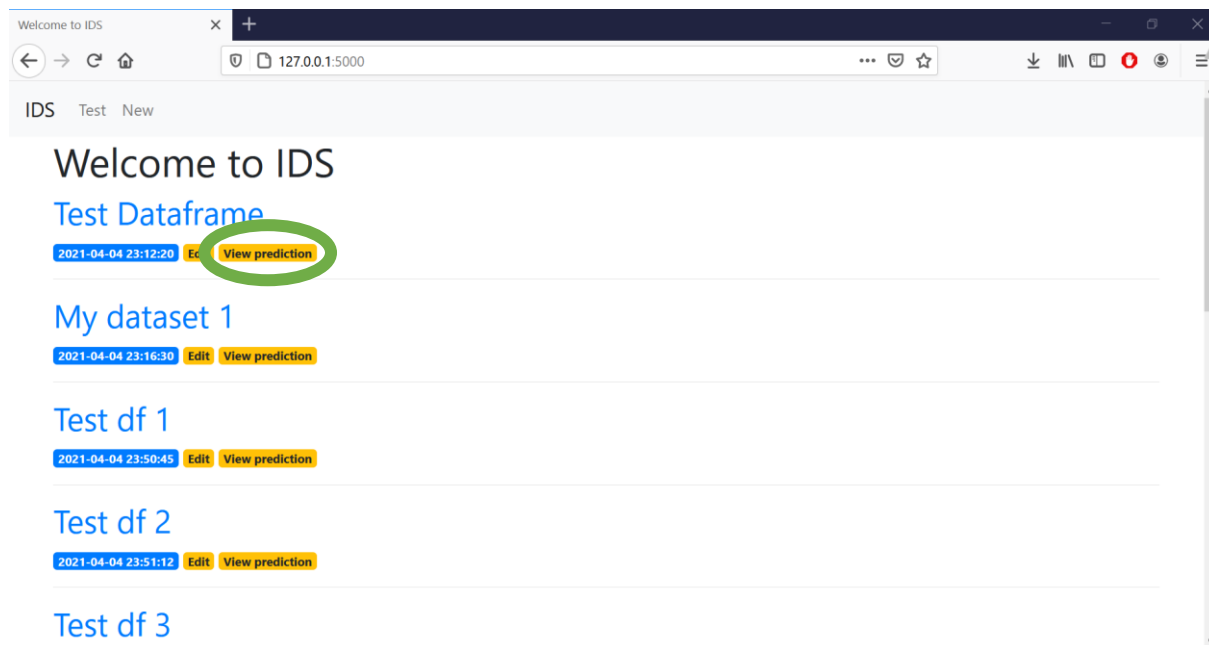


The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/test'. The page title is 'Test a dataset'. Below the title, there is a 'Title' label and a text input field containing the word 'Title'. Underneath, there is a label 'Choisir un fichier CSV' and a file selection area. This area includes a button labeled 'Parcourir...' and the text 'Aucun fichier sélectionné.'. At the bottom of the form, there is a blue button labeled 'Import'.

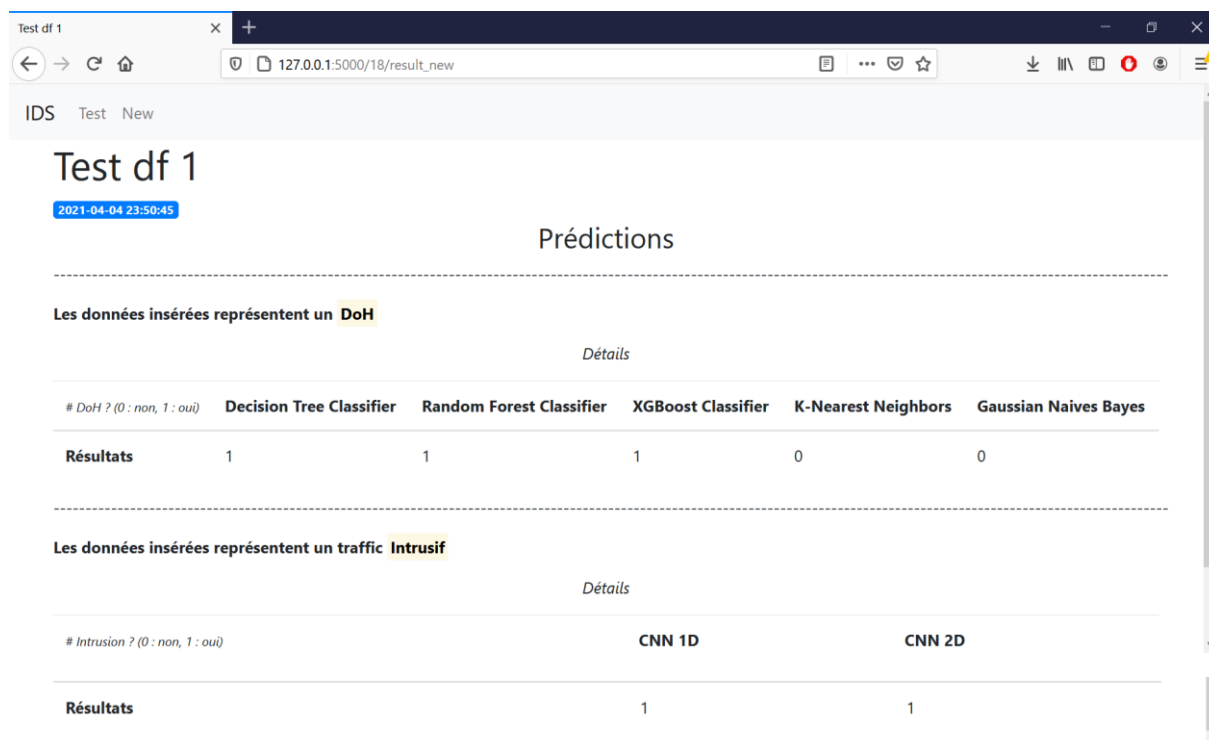
Ici il peut charger un fichier CSV qu'il a en local. Ici on récupère le nom du fichier puis on fait exactement le même traitement que précédemment sur les données. Une fois fait, l'utilisateur est redirigé sur la page lui présentant les prédictions.

6. Prédictions

Les prédictions sont donc visibles lorsque l'utilisateur soumet un fichier CSV ou qu'il insère de nouvelles données. De plus, l'utilisateur peut les consulter à tout moment lorsqu'il appuie sur le bouton 'View Predictions' sur la page d'accueil :

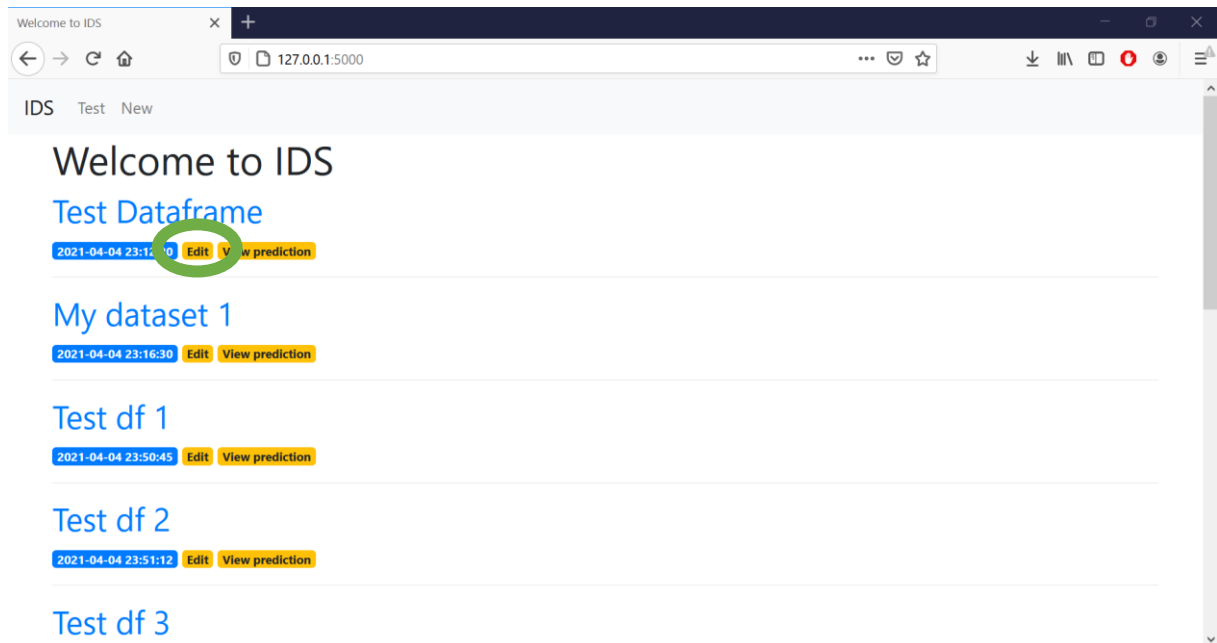


Ces trois actions amènent donc l'utilisateur sur cette page :

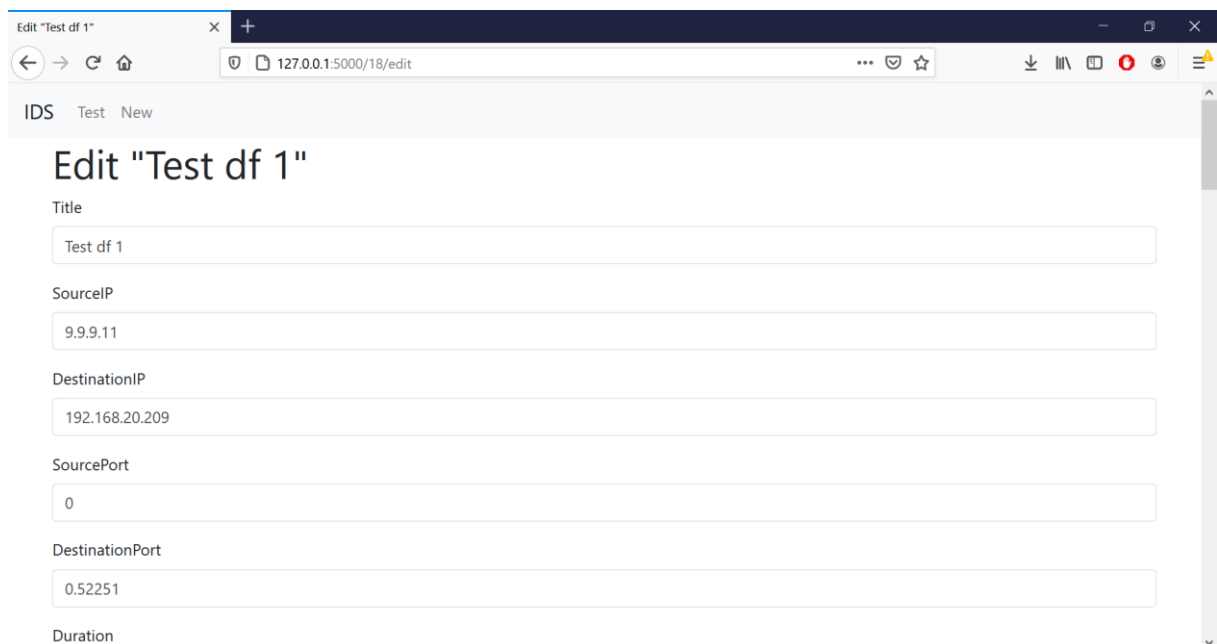


7. Edit / Delete

Enfin l'utilisateur peut également modifier ses données ou les supprimer en cliquant sur le bouton 'Edit' sur la page d'accueil :



Lorsqu'il clique dessus, il obtient un formulaire comme suit :

A screenshot of a web browser displaying the 'Edit "Test df 1"' form. The browser's address bar shows '127.0.0.1:5000/18/edit'. The form has a header with 'IDS', 'Test', and 'New' links. Below the header, there's a title 'Edit "Test df 1"'. The form contains several input fields: 'Title' (with 'Test df 1' entered), 'SourceIP' (with '9.9.9.11' entered), 'DestinationIP' (with '192.168.20.209' entered), 'SourcePort' (with '0' entered), 'DestinationPort' (with '0.52251' entered), and 'Duration' (empty). Each input field is preceded by its label.

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/18/edit'. The page title is 'Edit "Test df 1"'. The form contains the following fields and values:

Field Name	Value
ResponseTimeTimeMedian	0.0
ResponseTimeTimeMode	0.02173
ResponseTimeTimeSkewFromMedian	0.78722
ResponseTimeTimeSkewFromMode	0.59201
ResponseTimeTimeCoefficientofVariation	0.0119

At the bottom of the form, there are two buttons: a blue 'Submit' button and a red 'Delete' button.

Si l'utilisateur effectue des modifications et qu'il les soumet, alors la base de données sera mise à jour, de même pour le bouton 'Delete', les données seront supprimées définitivement de la base de données.

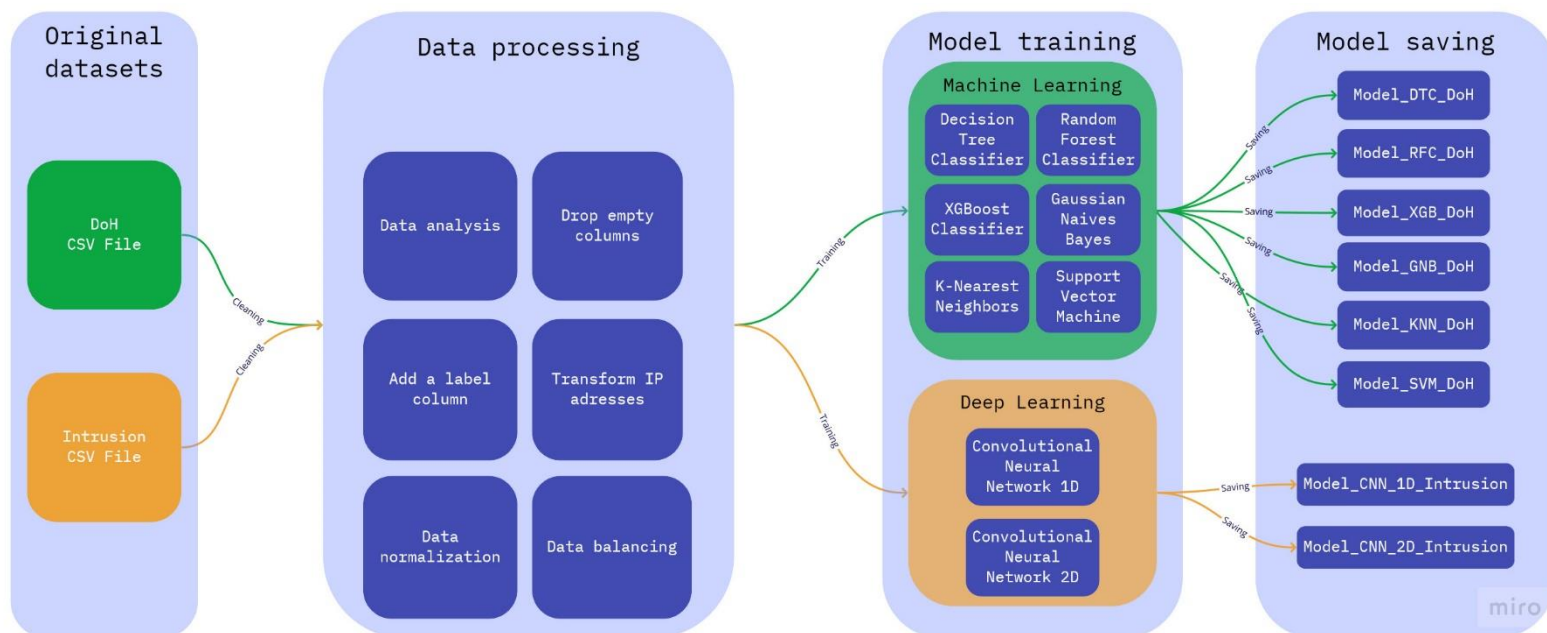
Conclusion

Travail réalisé

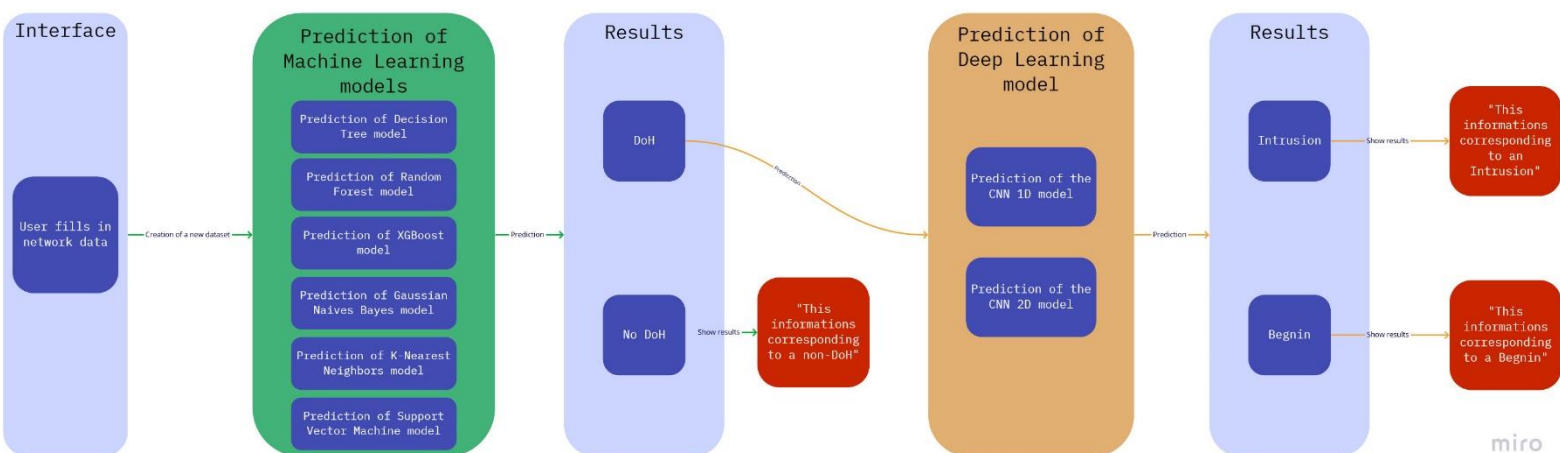
Pour résumer, nous recevons des datasets contenant des informations de connexion via le protocole DoH ainsi que des intrusions. Nous les traitons pour injecter, d'une part les données DoH dans nos modèles de Machine Learning et d'autre part les données d'intrusions dans nos modèles de Deep Learning. Ensuite, nous sauvegardons l'entraînement de ces modèles pour ensuite effectuer une prédiction sur de nouvelles données rentrées par l'utilisateur, soit via l'interface graphique de notre application locale, soit via l'interface de notre application web. Pour les deux applications, nous affichons à l'utilisateur le résultat des prédictions pour chaque modèle.

Voici deux diagrammes représentant l'architecture de notre projet et son fonctionnement.

Back-end



Front-end

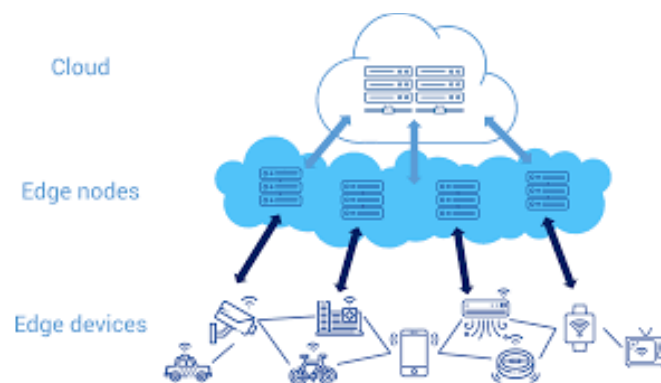


Perspectives

La suite logique du projet est de proposer notre solution aux entreprises. Nous avons créé deux applications qui permettent une utilisation différente de notre produit, et qui ne cible pas les mêmes types d'utilisateurs. D'une part, l'application en local cible des particuliers qui souhaiteraient dans un premier temps vérifier des informations de connexions issues d'internet. L'objectif serait à terme d'inclure dans cette application un système de monitoring réseau qui permettrait de retirer des informations de connexion réseau. Cela impliquerait de simplifier notre dataset d'entraînement pour pouvoir ne laisser que des features que le système de tracking de réseau serait capable d'identifier. Notre application deviendrait alors un système de sécurité temps réel contre les attaques sur le protocole DoH.

En ce qui concerne l'application web, le type de client visé serait des entreprises qui souhaitent vérifier les informations de connexion sur leurs serveurs. Cela nécessiterait dans un premier temps qu'une entreprise nous fournisse les données qu'elles souhaiteraient examiner. Cette application pourrait dans un second temps être une application d'analyse réseau, avec un suivi d'analyses poussées de leur flux réseau.

Dans un deuxième temps, notre projet est adapté aux nouvelles technologies du cloud comme le Edge computing qui est une méthode d'optimisation consistant à traiter les données à la périphérie du réseau, près de la source des données.



Cela permettrait de décentraliser les serveurs de base de données tout en laissant un temps de réponse adapté à la demande d'un client grâce aux Edge nodes, plus proche de l'utilisateur.

Le Federated Learning est aussi une application qui s'ancre parfaitement au sein de notre projet. Avec le Federated Learning, l'apprentissage des nouvelles données peut être effectué directement via le terminal d'un utilisateur. Seuls les paramètres et poids d'un modèle de réseau de neurones entraîné sur l'ordinateur d'un utilisateur sont transmis aux serveurs. Ces derniers centralisent les apprentissages au sein d'un réseau de neurones global. Ainsi, les données ne quittent pas le périphérique des utilisateurs.