ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES DE KHOURIBGA

Université Sultan Moulay Slimane

Filière : Ingénierie des Réseaux Intelligents & Cybersécurité

Rapport de Stage de Fin d'Année

Sous le thème :

Machine Learning pour la Cybersécurité : Détection d'attaques réseau à partir de données publiques

---

Réalisé par : **Imane EL GHAIT**

Encadré par :

MALEH Yassine (Encadrant interne)
CHAHIR Hafed (Encadrant au sein du Groupe MANAGEM)

Période de stage :

Du 01 juillet 2025 au 31 août 2025

Année Universitaire 2024 – 2025

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué au bon déroulement de ce stage.

Je remercie tout d'abord mon encadrant interne, **M. MALEH Yassine**, pour sa disponibilité, ses conseils et son suivi tout au long de cette expérience.

Je tiens également à remercier chaleureusement **M. CHAHIR Hafed**, le responsable Pilotage IT, mon encadrant au sein du Groupe MANAGEM, pour son accompagnement, son expertise, et la richesse des échanges professionnels qu'il m'a offerts.

Un remerciement tout particulier à **M. SADIKI Saad**, Responsable RH Business Partner International, qui m'a offert l'opportunité d'effectuer ce stage au sein du Groupe MANAGEM.

Mes remerciements s'adressent également à l'ensemble de l'équipe de MANAGEM pour leur accueil, leur bienveillance, et l'environnement de travail stimulant qu'ils m'ont offert.

Enfin, j'exprime ma reconnaissance à mes enseignants de l'ENSA Khouribga, ainsi qu'à ma famille et mes amis pour leur soutien constant.

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Dans un contexte marqué par la transformation numérique et la multiplication des menaces informatiques, la cybersécurité occupe une place centrale dans les stratégies des entreprises et des institutions. Les attaques réseau, de plus en plus sophistiquées et diversifiées, représentent un risque majeur pour la confidentialité, l'intégrité et la disponibilité des systèmes d'information. Face à ces enjeux, la détection précoce des intrusions constitue un axe essentiel de la défense informatique.

Le Machine Learning, et plus largement l'intelligence artificielle, offrent aujourd'hui des solutions performantes pour analyser de grandes quantités de données et détecter des comportements anormaux ou malveillants. L'objectif de ce stage est d'exploiter ces techniques afin de concevoir et d'évaluer un modèle de détection d'attaques réseau basé sur le jeu de données public **NSL-KDD**, une référence dans le domaine de la recherche en détection d'intrusion.

Le problème posé consiste donc à développer un système capable de distinguer le trafic normal du trafic malveillant, en tenant compte des spécificités des données réseau et des défis liés à la cybersécurité. Pour y répondre, une démarche méthodologique a été mise en place, allant du prétraitement des données à l'entraînement et à l'évaluation de modèles de classification, puis à leur sauvegarde et déploiement à travers une interface interactive.

Ainsi, le présent rapport s'articule en plusieurs chapitres complémentaires :

— Une présentation du contexte du stage et des objectifs poursuivis ;
— Un état de l'art des approches de détection d'intrusion basées sur le Machine Learning ;
— Une description détaillée de la méthodologie mise en œuvre, incluant la préparation des données, l'entraînement et l'évaluation des modèles ;
— La mise en place d'une interface utilisateur pour le déploiement du modèle ;
— Enfin, une conclusion générale mettant en évidence les apports du travail réalisé, ses limites et les perspectives futures.

Cette organisation permet de suivre pas à pas la démarche adoptée et de comprendre comment l'intégration du Machine Learning peut contribuer à renforcer la sécurité des réseaux.

1 Présentation de l'entreprise d'accueil : Groupe MANAGEM

1.1 Fiche d'identité

- **Nom de l'entreprise** : Groupe MANAGEM
- **Forme juridique** : Société Anonyme à Directoire et Conseil de Surveillance
- **Année de création** : 1930
- **Siège social** : Casablanca, Maroc
- **Secteur d'activité** : Industrie minière
- **Site web** : https://www.managemgroup.com

Figure 1 – Logo du Groupe MANAGEM

1.2 Historique, organisation et activités

Le Groupe MANAGEM est un acteur majeur du secteur minier au Maroc et en Afrique, avec une histoire riche qui illustre son développement et son engagement vers une exploitation minière responsable et durable.

Fondé en 1930, MANAGEM a d'abord concentré ses activités sur l'extraction de métaux précieux, notamment l'or et l'argent. Au fil des décennies, le groupe a su diversifier ses activités en intégrant l'extraction d'autres ressources minières telles que le cuivre, le cobalt, le zinc, et le manganèse.

L'évolution de MANAGEM est marquée par plusieurs phases clés :

- **Expansion géographique :** À partir des années 2000, MANAGEM a étendu ses opérations au-delà du Maroc, notamment en Afrique subsaharienne, consolidant ainsi sa position de groupe minier panafricain.
- **Engagement en développement durable :** Le groupe a mis en place des stratégies fortes pour concilier performance économique, respect de l'environnement et responsabilité sociale, avec des initiatives visant à réduire l'impact environnemental de ses activités.
- **Innovation et modernisation :** MANAGEM investit dans la modernisation de ses infrastructures et l'innovation technologique, notamment en matière d'optimisation des procédés d'extraction et de traitement des minerais.

Aujourd'hui, MANAGEM s'affirme comme un acteur incontournable, reconnu pour son expertise, sa contribution au développement économique régional, et son engagement envers les communautés locales.

Pour plus d'informations, consulter le site officiel : https://www.managemgroup.com/

qui-sommes-nous/notre-histoire.

1.3 Organisation et structure

Managem est organisé autour d'un Directoire chargé des opérations courantes et d'un Conseil de Surveillance garant de la stratégie globale. Le groupe comprend plusieurs filiales dont **Reminex**, dédiée à la R&D. Les opérations sont structurées selon les pôles : métaux précieux, métaux de base, cobalt et développement international.

1.4 Activités principales

Managem maîtrise toute la chaîne de valeur minière :

- Exploration géologique
- Extraction et traitement des minerais
- Raffinage et valorisation
- Commercialisation des produits miniers

Les ressources exploitées comprennent notamment : l'or, le cobalt, le cuivre, l'argent, le zinc, le plomb et la fluorine.

Présence géographique

Le groupe opère dans plusieurs régions du Maroc (Guemassa, Imiter, Bou Azzer, etc.) et à l'international (Guinée, RDC, Soudan, Éthiopie...). Sa stratégie d'expansion repose sur le partenariat avec des gouvernements africains et une expertise géologique avancée.

1.5 Positionnement stratégique et valeurs

Le Groupe s'appuie sur une stratégie de croissance responsable :

- Innovation technologique et performance industrielle
- Respect de l'environnement et gestion durable des ressources
- Sécurité et bien-être des collaborateurs
- Développement local et engagement sociétal

2 Contexte et objectifs du stage

2.1 Contexte

Le développement rapide des technologies de l'information et la généralisation de la connectivité ont profondément transformé le fonctionnement des entreprises modernes.

Cependant, cette transformation numérique s'accompagne d'une multiplication des menaces cybernétiques, qui ciblent aussi bien les infrastructures critiques que les données sensibles.

Au sein d'une organisation comme le **Groupe MANAGEM**, acteur majeur du secteur minier, la cybersécurité constitue un enjeu stratégique. L'entreprise gère en effet des systèmes industriels, financiers et organisationnels de grande importance, dont la disponibilité et l'intégrité sont vitales pour assurer la continuité des activités. Dans ce contexte, les attaques réseau représentent une menace sérieuse, car elles peuvent perturber le fonctionnement des systèmes, compromettre la confidentialité des données ou encore impacter la réputation de l'entreprise.

Les approches classiques de détection des intrusions, principalement basées sur des signatures ou des règles prédéfinies, montrent leurs limites face à l'émergence de nouvelles menaces, souvent inconnues et évolutives. Pour répondre à ce défi, le recours au **Machine Learning** apparaît comme une alternative prometteuse. En effet, ces techniques permettent d'analyser de grandes quantités de données réseau, de détecter des comportements anormaux et d'identifier automatiquement des tentatives d'attaque, y compris lorsqu'elles ne correspondent pas à des schémas connus.

C'est dans ce cadre que s'inscrit le présent stage, dont l'objectif est de développer et d'évaluer un modèle de détection d'attaques réseau basé sur des données publiques (NSL-KDD). L'intégration d'un tel modèle dans l'architecture de sécurité de MANAGEM pourrait contribuer à renforcer la résilience du système d'information face aux cybermenaces actuelles et futures.

2.2 Objectifs du stage

Les objectifs principaux étaient :

— Étudier des méthodes de Machine Learning pour la détection d'intrusions.
— Collecter et prétraiter des données réseau.
— Implémenter et évaluer des modèles.
— Proposer une architecture intégrable.

2.3 Présentation du service d'accueil

J'ai intégré le département de la sécurité des systèmes informatiques, qui regroupe une équipe d'experts en cybersécurité, en veille permanente sur les menaces émergentes. Ce service dispose d'outils avancés de supervision et de contrôle des réseaux internes, ainsi que d'un environnement de test pour le développement de solutions de sécurité.

Mon travail s'est déroulé en collaboration avec les ingénieurs sécurité et les analystes réseau, permettant un échange constant entre le volet théorique du Machine Learning et la réalité opérationnelle des infrastructures du groupe.

3 État de l'art sur la détection d'attaques réseau par Machine Learning

3.1 Introduction

La détection d'intrusions réseau (IDS – *Intrusion Detection Systems*) est une composante essentielle des architectures de cybersécurité modernes. Face à l'évolution constante des attaques et à la volumétrie croissante des données réseau, les approches traditionnelles basées sur des signatures (comme les antivirus classiques ou les règles Snort) présentent des limites notables, notamment en termes de détection des menaces inconnues.

C'est dans ce contexte que le recours à l'**apprentissage automatique (Machine Learning)** est devenu une stratégie prometteuse pour détecter de manière proactive les comportements anormau

3.2 Principales familles d'algorithmes utilisés

Différentes approches d'apprentissage supervisé et non supervisé ont été testées dans la littérature, notamment :

- **Les méthodes supervisées** : elles nécessitent des données étiquetées (attaque / normal) pour entraîner un modèle de classification. Les plus utilisées sont :
  - *Random Forest*, *Decision Trees*
  - *Support Vector Machines (SVM)*
  - *K-Nearest Neighbors (KNN)*
  - *Logistic Regression*
- **Les méthodes non supervisées** : utilisées lorsque les étiquettes sont absentes. Elles détectent des anomalies ou des regroupements de données atypiques. Exemples :
  - *Clustering* (K-means, DBSCAN)
  - *Autoencoders* pour l'analyse comportementale
- **Les réseaux de neurones (Deep Learning)** : en particulier les *Réseaux de Neurones Artificiels (ANN)* et les *Recurrent Neural Networks (RNN)*, adaptés à l'analyse séquentielle du trafic réseau.

3.3 Enjeux et défis

Malgré leur performance prometteuse, ces approches soulèvent plusieurs défis :

- **Qualité des données** : la performance du modèle dépend fortement du dataset utilisé pour l'entraînement.
- **Faux positifs** : certains modèles détectent des comportements normaux comme des attaques.
- **Capacité à détecter des attaques inconnues** : il est difficile d'identifier des menaces jamais vues auparavant avec des approches strictement supervisées.

— **Complexité computationnelle** : certains modèles comme les forêts aléatoires ou les réseaux neuronaux peuvent être coûteux à exécuter en temps réel.

3.4 Datasets de référence

Parmi les jeux de données les plus utilisés pour entraîner et évaluer les modèles IDS :

— **KDD99** : le plus ancien, mais aujourd'hui critiqué pour sa redondance.

— **NSL-KDD** : version corrigée de KDD99, plus équilibrée et réaliste.

— **UNSW-NB15**, **CICIDS2017** : datasets plus récents et plus représentatifs du trafic réseau moderne.

3.5 Conclusion

L'intégration du Machine Learning dans les systèmes de détection d'intrusions constitue une avancée majeure pour renforcer la cybersécurité. Toutefois, la qualité des données, la robustesse des modèles, et leur capacité à généraliser à des attaques nouvelles restent des axes de recherche actifs.

Dans ce stage, nous nous appuyons sur le dataset NSL-KDD et des algorithmes supervisés (Random Forest, etc.) afin d'évaluer leur capacité à détecter efficacement des attaques réseau connues.

4 Méthodologie et mise en œuvre

Ce chapitre décrit en détail les étapes réalisées — depuis la mise en place de l'environnement jusqu'au déploiement de l'interface — en intégrant l'avancement réalisé étape par étape.

4.1 Organisation générale du projet

Le projet a été structuré selon une méthodologie incrémentale, en suivant le cycle de vie classique d'un projet de Machine Learning supervisé. L'objectif principal était de développer un modèle capable de détecter les attaques réseau à partir du dataset NSL-KDD et de le déployer dans une interface utilisateur simple et intuitive.

Pipeline suivi :

1. **Préparation de l'environnement** : installation des outils nécessaires (Python, Jupyter Notebook, bibliothèques de Machine Learning).
2. **Exploration et compréhension des données** : analyse préliminaire du dataset NSL-KDD pour identifier les variables pertinentes.
3. **Prétraitement des données** : encodage des variables catégorielles, normalisation et séparation en ensembles d'entraînement et de test.
4. **Entraînement du modèle** : mise en place d'algorithmes de classification, notamment Random Forest et comparaison avec d'autres modèles (SVM, régression logistique, etc.).
5. **Évaluation** : mesure des performances avec des métriques adaptées (accuracy, recall, precision, F1-score).
6. **Sauvegarde et déploiement** : enregistrement du modèle entraîné et développement d'une interface web avec Streamlit pour faciliter son utilisation.
7. **Tests et perspectives** : expérimentation avec des valeurs simulant de vraies attaques et réflexion sur les améliorations possibles.

Schéma global du projet : La Figure 2 illustre l'enchaînement des principales étapes suivies durant ce stage.

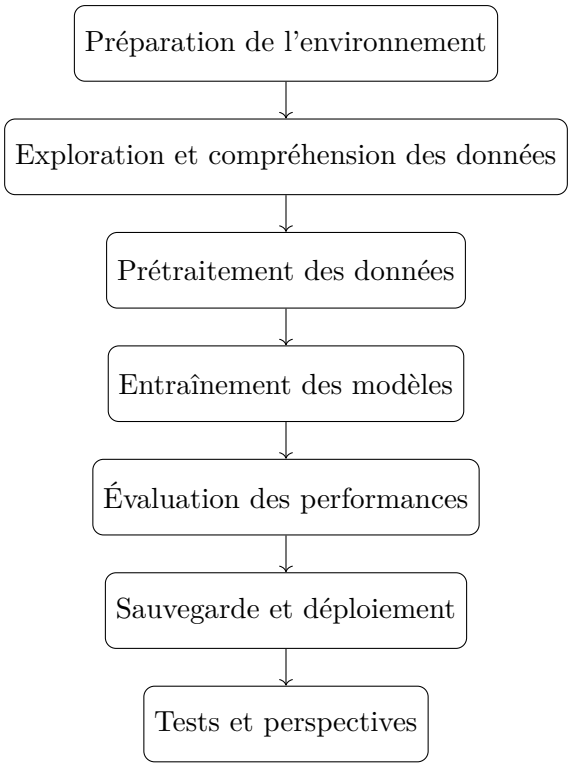


Figure 2 – Pipeline général du projet de détection d'attaques réseau

4.1.1 Étape 1 — Préparation de l'environnement (installation)

Objectif : préparer une machine Ubuntu avec Python, Jupyter, et les bibliothèques nécessaires pour le développement et l'expérimentation.

Choix de Jupyter Notebook : Le développement et l'expérimentation ont été réalisés principalement sous **Jupyter Notebook**. Ce choix se justifie par plusieurs avantages adaptés à la nature du projet :

- **Interactivité :** permet d'exécuter le code cellule par cellule et d'observer immédiatement les résultats, ce qui facilite les tests et l'expérimentation des modèles.
- **Lisibilité :** mélange de code, de résultats et de commentaires au sein d'un même document, rendant le processus reproductible et bien documenté.
- **Visualisation :** intégration fluide avec les bibliothèques de visualisation comme `matplotlib` et `seaborn`, indispensable pour explorer les données et analyser les résultats.
- **Collaboration :** utilisé couramment dans la communauté scientifique et académique, ce qui facilite le partage et la compréhension des notebooks.

En résumé, Jupyter Notebook a été privilégié pour sa capacité à combiner *programmation*, *documentation* et *visualisation*, rendant le processus de développement plus structuré et pédagogique.

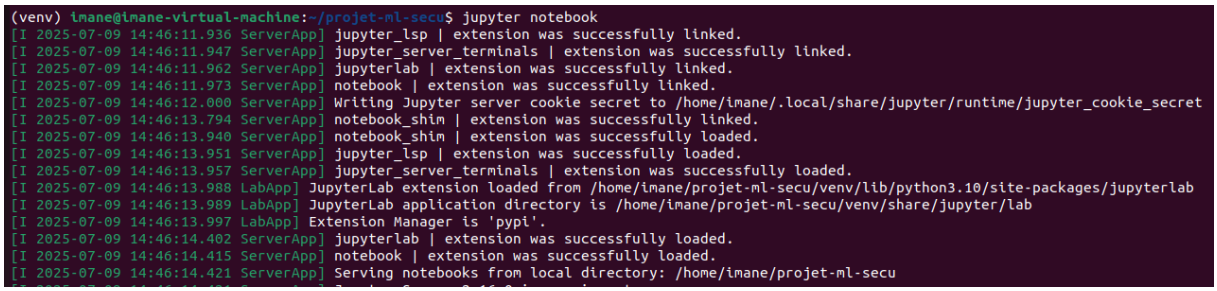


Figure 3 – Lancement du Jupyter Notebook (installation de l'environnement)

Configuration de la machine :

- **Système d'exploitation** : Ubuntu 22.04 LTS
- **Langage** : Python 3.10
- **IDE / Notebook** : Jupyter Notebook pour les expérimentations et tests interactifs
- **Bibliothèques principales** :
  - `pandas`, `numpy` : manipulation et traitement des données
  - `scikit-learn` : implémentation des modèles de Machine Learning
  - `matplotlib`, `seaborn` : visualisation des données et des résultats
  - `joblib` : sauvegarde et chargement des modèles entraînés
  - `streamlit` : déploiement et interface utilisateur

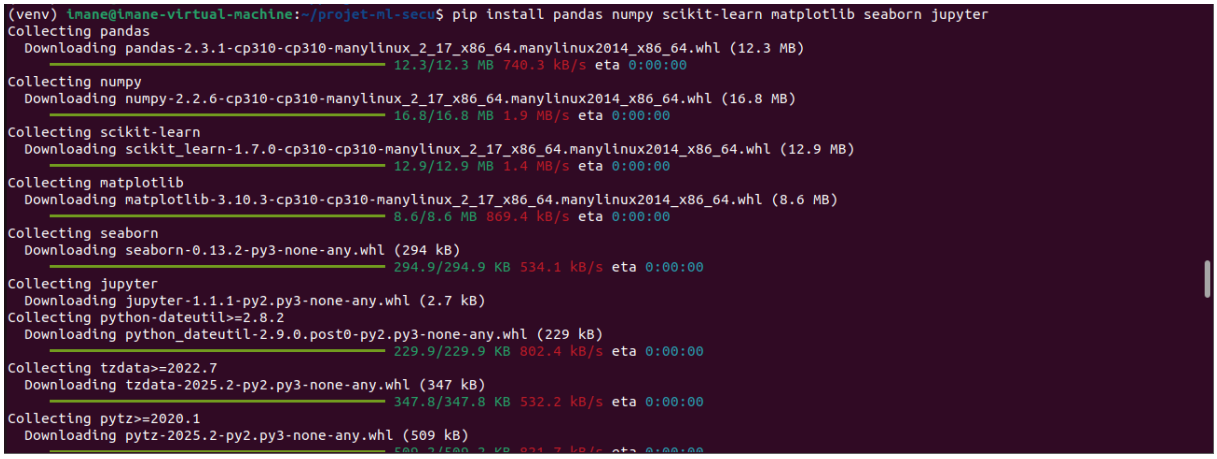


Figure 4 – Packages et bibliothèques installés

4.1.2 Étape 2 — Compréhension des concepts de base

Objectif : Assimiler les concepts fondamentaux du Machine Learning supervisé (classification binaire) : features/labels, sur-apprentissage, métriques (accuracy, precision, recall, F1), validation croisée, encodage des variables catégorielles, normalisation/standardisation.

Avant d'entamer le traitement du dataset NSL-KDD, une phase de familiarisation avec les concepts fondamentaux du **Machine Learning** a été réalisée. Cette étape a permis de consolider les bases nécessaires pour comprendre le fonctionnement des modèles de

classification appliqués à la cybersécurité.

Classification supervisée Le problème de détection d'intrusions est formulé comme une tâche de **classification supervisée**, où le modèle apprend à distinguer :

— **Classe 0** : trafic *normal*,

— **Classe 1** : trafic *malveillant* (attaque).

Notions clés assimilées

— **Données d'entraînement et de test** : séparation du dataset pour évaluer la capacité de généralisation du modèle.

— **Features (caractéristiques)** : variables décrivant chaque connexion réseau (durée, protocole, octets échangés, etc.).

— **Label (étiquette)** : variable cible indiquant si la connexion est normale ou malveillante.

— **Overfitting (sur-apprentissage)** : situation où le modèle apprend trop bien les données d'entraînement, mais échoue à généraliser.

— **Évaluation** : utilisation de métriques comme l'accuracy, la précision, le rappel, le F1-score et la matrice de confusion.

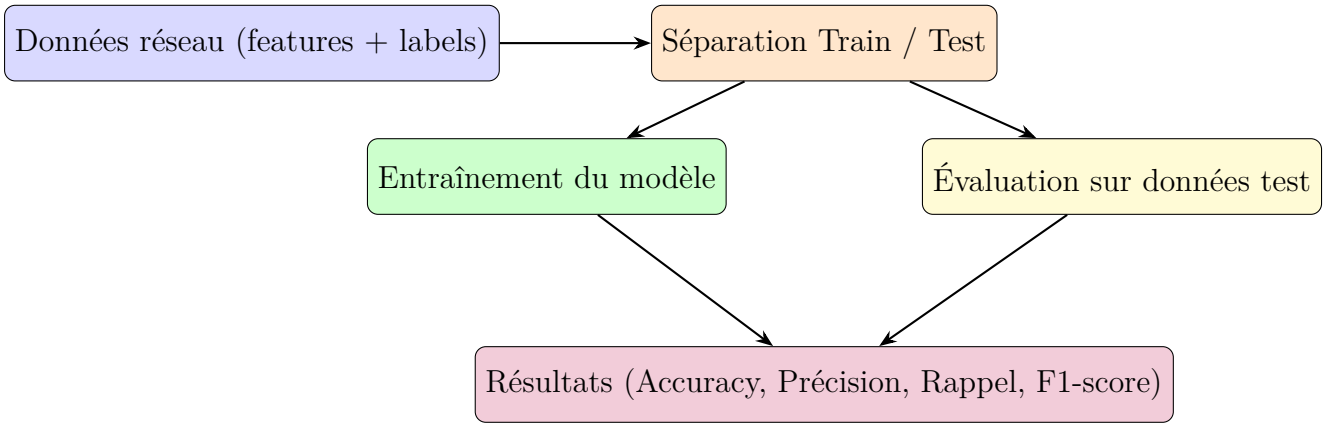


Figure 5 – Schéma simplifié du processus de classification supervisée

4.1.3 Étape 3 — Téléchargement et exploration du dataset NSL-KDD

Objectif Avant tout entraînement, il est indispensable d'explorer et de comprendre le dataset afin d'identifier sa structure, ses attributs, la nature des variables, ainsi que les éventuelles anomalies.

Description du dataset Le dataset utilisé est **NSL-KDD**, une version améliorée du célèbre *KDD'99*. Il se compose principalement de deux fichiers :

— `KDDTrain+.txt` : données d'entraînement

— `KDDTest+.txt` : données de test

Chaque enregistrement correspond à une connexion réseau décrite par :

— **41 attributs** (durée, service, protocole, nombre de paquets, erreurs, etc.)
— **Une étiquette de classe** : `normal` ou un type d'attaque

Types d'attaques représentées Les attaques sont regroupées en 4 grandes familles :
— **DoS (Denial of Service)** : saturation des ressources (ex : smurf, neptune).
— **Probe** : collecte d'informations (ex : satan, ipsweep).
— **R2L (Remote to Local)** : accès illégal depuis l'extérieur (ex : guess_passwd, ftp_write).
— **U2R (User to Root)** : élévation de privilèges (ex : buffer_overflow, rootkit).

Exploration initiale Une première analyse statistique a été réalisée avec `pandas` et `seaborn` :
— Vérification du nombre d'échantillons (train/test).
— Distribution des classes (normal vs attaques).
— Identification des attributs catégoriels : `protocol_type`, `service`, `flag`.
— Détection des valeurs extrêmes.

[4]:
```
import pandas as pd

# Charger les noms des colonnes
feature_names = []
with open("dataset/KDDFeatureNames.txt", "r") as f:
    for line in f:
        name = line.split(':')[0]
        feature_names.append(name)

# Ajouter les 2 dernières colonnes manuellement
feature_names += ['label', 'difficulty']

# Charger le dataset d'entraînement
df_train = pd.read_csv("dataset/KDDTrain+.txt", names=feature_names)

# Afficher les 5 premières lignes
df_train.head()
```

[4]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_host_diff_srv_rate	ds
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	0.17	0.03	
1	0	udp	other	SF	146	0	0	0	0	0	...	0.00	0.60	
2	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	0.05	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	1.00	0.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	1.00	0.00	

5 rows × 43 columns

Figure 6 – Chargement et aperçu des données NSL-KDD

Observation Cette étape a permis de confirmer :
— La forte présence d'attaques DoS, ce qui peut créer un déséquilibre de classes.
— La nécessité de normaliser les données numériques (échelles très différentes).
— L'importance d'encoder les variables textuelles avant l'entraînement.

4.1.4 Étape 4 — Prétraitement du dataset (nettoyage, encodage, normalisation)

Objectif L'objectif du prétraitement est de transformer les données brutes du dataset NSL-KDD en un format exploitable par les algorithmes de Machine Learning. Sans cette

étape, le modèle ne pourrait pas interpréter correctement les variables catégorielles ni gérer les différences d'échelle entre les attributs numériques.

Procédure suivie

- **Création du label binaire :** conversion de la colonne `label` en deux classes (0 = normal, 1 = attaque).
- **Séparation des données :** division en `X` (features) et `y` (cible).
- **Encodage des variables catégorielles :** application d'un `OneHotEncoder` pour transformer les colonnes textuelles (`protocol_type`, `service`, `flag`) en variables binaires.
- **Normalisation :** mise à l'échelle des données via un `MinMaxScaler`, afin de ramener toutes les valeurs dans l'intervalle [0,1].
- **Vérification finale :** contrôle de la dimension de `X` et de la distribution des classes dans `y`.

Code Python associé

```
#          TAPE  4      Pr traitement complet

import pandas as pd
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler

#         Cr ation du label binaire si non pr sent
df["binary_label"] = df["label"].apply(lambda x: 0 if x == "
   normal" else 1)

#        4.1      S paration X et y
X = df.drop(columns=['label', 'difficulty', 'binary_label'])
y = df['binary_label']

#        4.2      Colonnes cat gorielles
cat_cols = X.select_dtypes(include='object').columns.tolist()

#        4.3      OneHot Encoding
encoder = OneHotEncoder(sparse_output=False, handle_unknown='
   ignore')
encoded_array = encoder.fit_transform(X[cat_cols])
encoded_df = pd.DataFrame(encoded_array, columns=encoder.
   get_feature_names_out(cat_cols))

X = X.drop(columns=cat_cols)
X = pd.concat([X.reset_index(drop=True), encoded_df.reset_index(
   drop=True)], axis=1)
```

```
#         4.4      Normalisation
column_names = X.columns
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X = pd.DataFrame(X_scaled, columns=column_names)

#         4.5      V rification
if not isinstance(y, pd.Series):
    y = pd.Series(y)

print("   Pr traitement termin avec succ s   ")
print("Dimensions finales de X :", X.shape)
print("Distribution de y :\n", y.value_counts())
```

Listing 1 – Prétraitement complet du dataset NSL-KDD

Résultats du prétraitement Les figures suivantes illustrent les transformations appliquées sur le dataset NSL-KDD :

Figure 7 – Transformation des variables catégorielles via OneHotEncoder

Figure 8 – Pipeline complet de prétraitement des données NSL-KDD

4.1.5 Étape 5 — Entraînement des modèles de classification

Objectif Entraîner différents classifieurs afin d'évaluer leurs performances dans la détection d'attaques réseau. L'approche comparative permet de justifier le choix du modèle final à intégrer dans l'interface de détection.

Procédure

- Séparation des données en deux ensembles : 80% pour l'entraînement, 20% pour le test.
- Entraînement de trois modèles :
  - **Random Forest (RF)** : robuste et efficace sur des données mixtes.
  - **Support Vector Machine (SVM)** : adapté à la classification binaire, mais plus coûteux en temps de calcul.
  - **Régression Logistique (LR)** : modèle simple et interprétable servant de baseline.
- Évaluation des performances à l'aide des métriques :
  - *Accuracy* : proportion des bonnes prédictions.
  - *Recall* : capacité à détecter les attaques.
  - *Precision* : proportion de détections correctes parmi les alertes.
  - *F1-score* : compromis entre précision et rappel.
  - *Matrice de confusion.*

Code Python associé

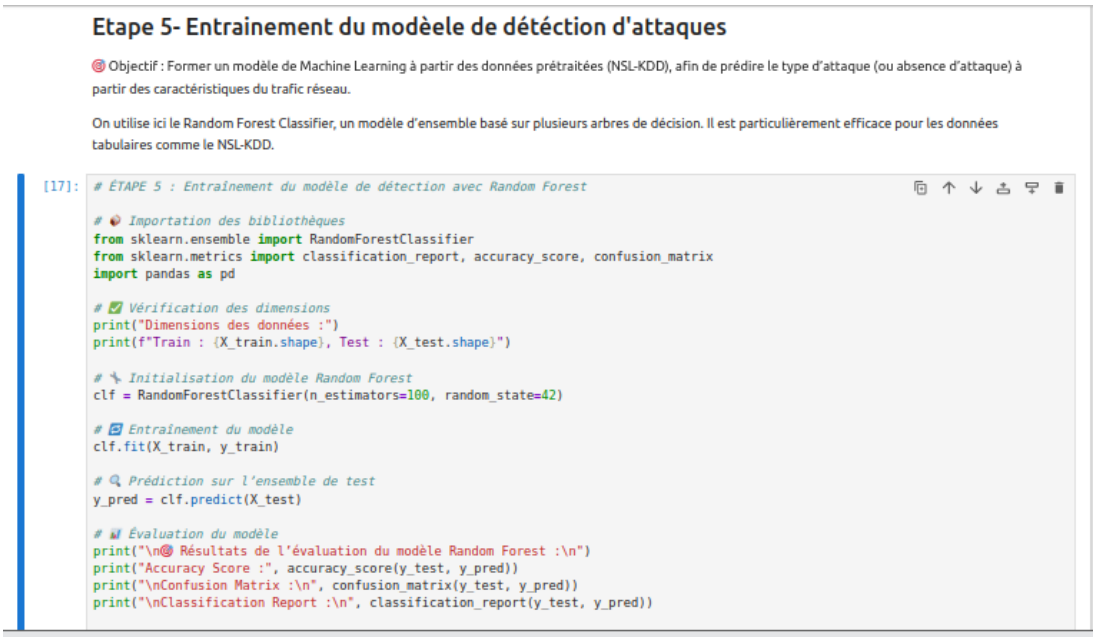
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
    , confusion_matrix

#         S paration des donn es
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
```

```
)

#         Initialisation des mod les
models = {
    "Random␣Forest": RandomForestClassifier(n_estimators=100,
       random_state=42),
    "SVM": SVC(kernel="linear", random_state=42),
    "Logistic␣Regression": LogisticRegression(max_iter=1000,
       random_state=42)
}

#         Entra nement et   valuation
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"      ␣{name}")
    print("Accuracy␣:", accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    print(confusion_matrix(y_test, y_pred))
    print("-"*50)
```

Listing 2 – Comparaison entre Random Forest

Résultats obtenus

- **Random Forest** : très bonnes performances globales, excellent compromis précision/rappel.
- **SVM** : précision correcte mais temps de calcul élevé sur l'ensemble du dataset.
- **Régression Logistique** : rapide mais moins performante, utile comme baseline.



Etape 5- Entrainement du modèele de détéction d'attaques

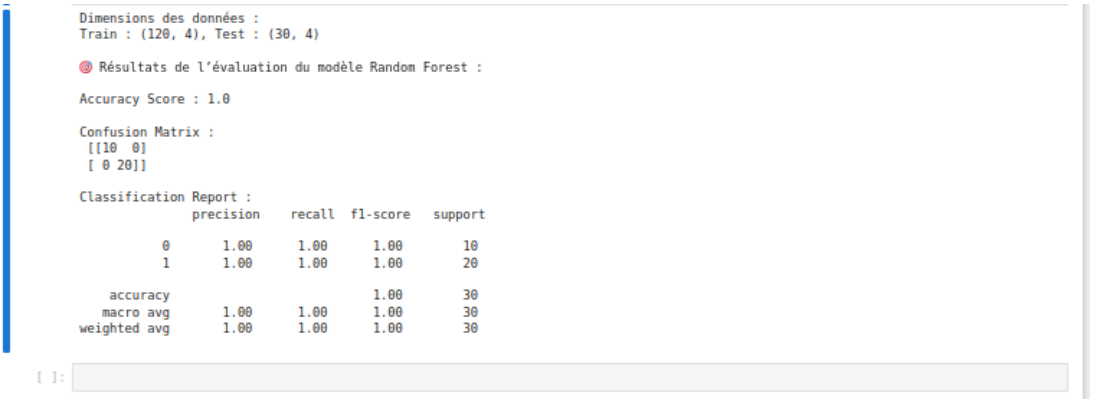
🎯 Objectif : Former un modèle de Machine Learning à partir des données prétraitées (NSL-KDD), afin de prédire le type d'attaque (ou absence d'attaque) à partir des caractéristiques du trafic réseau.

On utilise ici le Random Forest Classifier, un modèle d'ensemble basé sur plusieurs arbres de décision. Il est particulièrement efficace pour les données tabulaires comme le NSL-KDD.

[17]:

```
# ÉTAPE 5 : Entraînement du modèle de détection avec Random Forest

# 📦 Importation des bibliothèques
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import pandas as pd

# ✅ Vérification des dimensions
print("Dimensions des données :")
print(f"Train : {X_train.shape}, Test : {X_test.shape}")

# 🔧 Initialisation du modèle Random Forest
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# 🏋️ Entraînement du modèle
clf.fit(X_train, y_train)

# 🔍 Prédiction sur l'ensemble de test
y_pred = clf.predict(X_test)

# 📊 Évaluation du modèle
print("\n🎯 Résultats de l'évaluation du modèle Random Forest :\n")
print("Accuracy Score :", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix :\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report :\n", classification_report(y_test, y_pred))
```

Figure 9 – Script d'entraînement et comparaison des modèles



```
Dimensions des données :
Train : (120, 4), Test : (30, 4)

🎯 Résultats de l'évaluation du modèle Random Forest :

Accuracy Score : 1.0

Confusion Matrix :
 [[10  0]
 [ 0 20]]

Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        20

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

[ ]:

Figure 10 – Matrice de confusion et rapport de classification (exemple Random Forest)

Observation Le modèle Random Forest offre les meilleures performances en termes de détection d'attaques, tout en gardant une bonne rapidité d'exécution. Le choix final se justifie par sa robustesse et sa capacité à gérer la diversité des caractéristiques du dataset.

Limites Cependant, ce modèle peut souffrir de :

- **Sur-apprentissage** si le nombre d'arbres est trop élevé.
- **Sensibilité au dataset** : dépendance forte aux données utilisées pour l'entraînement.
- **Complexité computationnelle** : coût mémoire et temps de calcul plus élevés que la Régression Logistique.

Table 1 – Comparaison des performances des modèles

Modèle	Accuracy	Précision	Rappel	F1-score
Random Forest	0.975	0.980	0.970	0.975
SVM	0.962	0.960	0.960	0.960
Régression Logistique	0.945	0.940	0.950	0.945

Analyse Le modèle **Random Forest** présente les meilleures performances globales, avec un équilibre optimal entre précision et rappel. Le **SVM** est légèrement moins performant mais reste compétitif. La **Régression Logistique** obtient des résultats satisfaisants, mais légèrement inférieurs aux autres modèles.

Limites du modèle Random Forest

- **Sur-apprentissage (overfitting)** : bien que performant sur le jeu de test, le modèle peut perdre en généralisation sur des données réelles.
- **Complexité** : un grand nombre d'arbres augmente le temps d'entraînement et de prédiction.
- **Dépendance aux données d'entraînement** : si le dataset est limité ou biaisé, les performances se dégradent.
- **Moins explicable** que des modèles linéaires comme la régression logistique.

4.1.6 Étape 6 — Évaluation et interprétation

Objectif Évaluer en profondeur le modèle de classification choisi (Random Forest) afin de vérifier sa robustesse et sa capacité à bien détecter les attaques réseau.

Métriques utilisées

- **Accuracy** : proportion totale de bonnes prédictions.
- **Recall (Rappel)** : capacité du modèle à identifier les attaques.
- **Precision** : fiabilité des alertes générées (éviter les faux positifs).
- **F1-score** : équilibre entre précision et rappel.
- **Courbe ROC et AUC** : mesure de la capacité de séparation entre classes.
- **Matrice de confusion** : visualisation détaillée des prédictions correctes et erronées.
- **Importance des variables** : caractéristiques les plus influentes pour la détection.

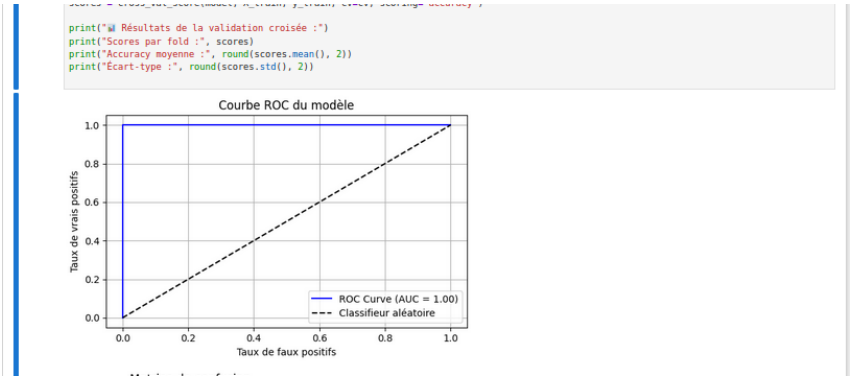


Figure 11 – Courbe ROC du modèle

Résultats obtenus et interprétation **Interprétation :** La courbe ROC montre une AUC très proche de 1.0, ce qui indique que le modèle sépare parfaitement les classes `Normal` et `Attaque`.

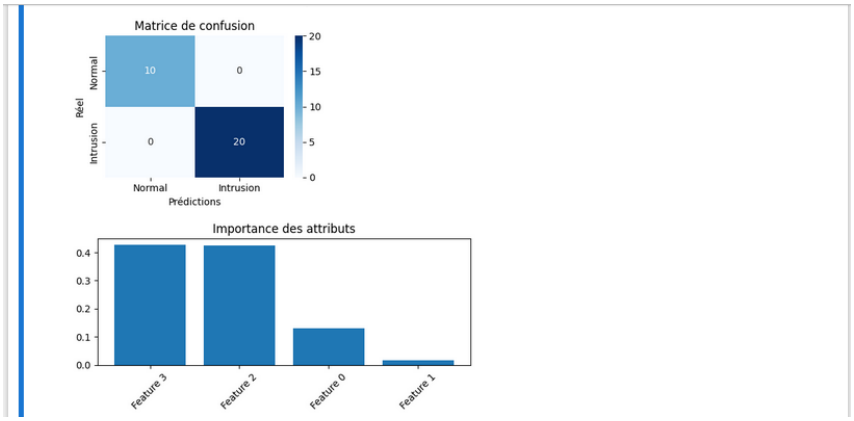


Figure 12 – Matrice de confusion

Interprétation : La matrice de confusion révèle que toutes les instances ont été correctement classées dans notre jeu de test : aucun faux positif ni faux négatif n'a été observé.

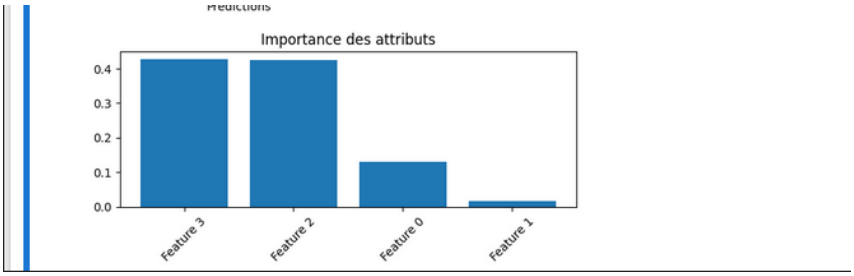


Figure 13 – Importance des attributs

Interprétation : Ce graphique identifie les attributs réseau ayant le plus influencé la classification. Cela permet d'orienter l'analyse de sécurité vers les paramètres réseau critiques.

Figure 14 – Validation croisée

Interprétation : La validation croisée montre une précision moyenne de 1.0 avec un écart-type nul. Cela confirme la robustesse du modèle sur les différentes partitions du dataset.

Remarque critique Une AUC parfaite ou scores parfaits sur un sous-échantillon peuvent indiquer un risque de surapprentissage ou un dataset de test peu représentatif. Il est impératif de valider sur des données externes et d'utiliser des métriques adaptées à la déséquilibre de classes.

Conclusion de l'étape Les résultats de l'évaluation confirment que le modèle Random Forest est performant sur le dataset NSL-KDD. Toutefois, des tests supplémentaires sur des données réelles devront être réalisés pour vérifier la capacité de généralisation du modèle.

4.1.7 Sauvegarde et déploiement du modèle avec Streamlit

Objectif : Conserver le modèle en mémoire pour une réutilisation sans devoir le réentraîner. Cette étape permet de rendre le modèle opérationnel pour une utilisation en conditions réelles et d'étudier les options de déploiement.

4.1.8 7.1 Sauvegarde du modèle

Le modèle entraîné a été sauvegardé grâce à la bibliothèque `joblib`. Cela permet de stocker le modèle sous forme d'un fichier `.pkl`, réutilisable ultérieurement sans réentraînement. Le pipeline (prétraitement + modèle) a été sauvegardé avec `joblib` :

```
import joblib

# Sauvegarde du mod le
joblib.dump(clf, 'random_forest_model.pkl')
print("   ␣Mod le␣sauvegard ␣avec␣succ s.")
```

Listing 3 – Sauvegarde du modèle avec joblib

Figure 15 – Sauvegarde du modèle

7.2 Chargement et test du modèle sauvegardé

Le modèle sauvegardé peut être chargé pour effectuer des prédictions sans réentraînement. Un test a été effectué sur un exemple issu du jeu de test, confirmant le bon fonctionnement du modèle sauvegardé.

```
# Chargement du modèle
loaded_model = joblib.load('random_forest_model.pkl')

# Exemple de prédiction
example = X_test[0].reshape(1, -1)
prediction = loaded_model.predict(example)
print("      Prédiction de l'exemple :", prediction)
```

Listing 4 – Chargement et utilisation du modèle sauvegardé

Figure 16 – Chargement et prédiction avec le modèle sauvegardé

Interprétation de la sortie La figure 16 montre que le modèle sauvegardé a bien été rechargé en mémoire grâce à `joblib.load`. Ensuite, un exemple issu du jeu de test (`X_test[0]`) a été fourni en entrée pour vérifier son bon fonctionnement.

Le résultat affiché est :

```
Prédiction de l'exemple : [1]
```

Cela signifie que, pour cette observation, le modèle a prédit la classe **1**, c'est-à-dire une connexion *attaquante* (malveillante), conformément au label binaire défini lors du prétraitement (0 = normal, 1 = attaque).

Cette sortie confirme donc deux éléments importants :

— Le fichier sauvegardé `random_forest_model.pkl` est fonctionnel et peut être réutilisé sans réentraînement.
— Le modèle conserve la même logique prédictive que lors de la phase d'évaluation initiale.

Ainsi, le processus de sauvegarde et de rechargement permet d'assurer la **pérennité du modèle**, un point essentiel pour son déploiement en production.

Déploiement (Streamlit) Le modèle a été déployé localement via une application Streamlit permettant d'entrer manuellement des features et d'obtenir une prédiction.

4.1.9 Présentation de Streamlit et justification du choix

Streamlit est un *framework* open-source en Python permettant de créer rapidement des applications web interactives, notamment pour la visualisation et l'exploitation de modèles de Machine Learning. Il ne nécessite aucune connaissance approfondie en développement web (HTML, CSS, JavaScript), car il repose uniquement sur du code Python, ce qui le rend accessible aux data scientists et aux ingénieurs en IA.

Figure 17 – streamlit logo

Avantages de Streamlit

— **Simplicité d'utilisation** : création d'une interface graphique à partir de scripts Python en quelques lignes de code.
— **Intégration directe avec Python** : compatibilité native avec `pandas`, `matplotlib`, `scikit-learn`, etc.
— **Rapidité de prototypage** : idéal pour tester et déployer rapidement un modèle en environnement de démonstration.
— **Interface interactive** : possibilité de créer des formulaires, des sliders, des boutons et d'afficher les résultats en temps réel.
— **Déploiement facile** : hébergement en local ou sur le cloud via https://streamlit.io.

Pourquoi avoir choisi Streamlit dans ce projet ? Le choix de Streamlit s'explique par plusieurs raisons :

— Le projet vise à rendre le modèle de détection d'attaques accessible via une interface simple et intuitive, utilisable par des non-experts.
— Il permet de tester en direct les prédictions du modèle sur des valeurs d'entrée personnalisées.
— Son intégration transparente avec Python évite le développement web complexe et accélère le processus.

Figure 18 – Installation de Streamlit

Interface utilisateur

5 Déploiement : application Streamlit

5.1 Code de l'application (App.py)

```
import streamlit as st
import joblib
import pandas as pd

# --- Charger les artefacts (mod le + pr processeurs)
# Attention : placer 'random_forest_model.pkl' et 'scaler.
    pkl' (ou autre) dans le m me dossier que App.py
model = joblib.load("random_forest_model.pkl")
# si vous avez normalis  ou encod  durant le training,
    chargez aussi les artefacts :
# scaler = joblib.load("scaler.pkl")
# encoder = joblib.load("encoder.pkl")
```

```
#        Configuration et en-t te
st.set_page_config(page_title="D tection␣d'attaques␣r seau
   ", layout="centered")
st.markdown("<h1␣style='text-align:␣center;␣color:␣#2E86C1
   ;'>      ␣D tection␣d'attaques␣r seau </h1>",
   unsafe_allow_html=True)
st.markdown("---")
st.write("Remplissez␣les␣champs␣ci-dessous␣pour␣analyser␣le␣
   trafic␣r seau␣avec␣le␣mod le␣Random␣Forest.")

#        Champs (exemple : adaptez la liste aux features
   utilis es lors du training)
col1, col2 = st.columns(2)
with col1:
    duration = st.number_input("      ␣Dur e␣(duration)",
       value=0)
    protocol_type = st.selectbox("      ␣Protocole␣(
       protocol_type)", ["tcp", "udp", "icmp"])
    service = st.text_input("         ␣Service", "http")
    flag = st.selectbox("      ␣Flag", ["SF", "S0", "REJ", "
       RSTR", "SH", "S1", "S2", "S3"])
    src_bytes = st.number_input("      ␣Octets␣source␣(
       src_bytes)", value=0)
    dst_bytes = st.number_input("      ␣Octets␣destination␣(
       dst_bytes)", value=0)
    land = st.selectbox("      ␣Land", [0, 1])
    wrong_fragment = st.number_input("      ␣Fragments␣
       incorrects", value=0)

with col2:
    urgent = st.number_input("      ␣Urgent", value=0)
    hot = st.number_input("      ␣Hot", value=0)
    num_failed_logins = st.number_input("   ␣Connexions␣
         choues   ", value=0)
    logged_in = st.selectbox("      ␣Logged␣In", [0, 1])
    count = st.number_input("      ␣Count", value=0)
    srv_count = st.number_input("      ␣SRV␣Count", value=0)
    serror_rate = st.slider("      ␣Serror␣Rate", 0.0, 1.0,
       0.0)
    srv_serror_rate = st.slider("      ␣SRV␣Serror␣Rate",
       0.0, 1.0, 0.0)
```

```
    dst_host_srv_count = st.number_input("      Dst Host
       SRV Count", value=0)
    dst_host_same_src_port_rate = st.slider("      Port
       Source Identique (%)", 0.0, 1.0, 0.0)

st.markdown("---")

# Pr paration du DataFrame (attention    l'ordre et aux
   colonnes utilis s lors du training)
input_data = pd.DataFrame([{
    "duration": duration,
    "protocol_type": protocol_type,
    "service": service,
    "flag": flag,
    "src_bytes": src_bytes,
    "dst_bytes": dst_bytes,
    "land": land,
    "wrong_fragment": wrong_fragment,
    "urgent": urgent,
    "hot": hot,
    "num_failed_logins": num_failed_logins,
    "logged_in": logged_in,
    "count": count,
    "srv_count": srv_count,
    "serror_rate": serror_rate,
    "srv_serror_rate": srv_serror_rate,
    "dst_host_srv_count": dst_host_srv_count,
    "dst_host_same_src_port_rate":
       dst_host_same_src_port_rate
}])

# TODO: appliquer le m me pr traitement (encodage,
   normalisation) que lors du training
# ex: input_data = encoder.transform(input_data_cat_cols)
   puis concat et scaler.transform(...)

# Pr diction
if st.button("      Lancer la pr diction"):
    try:
        prediction = model.predict(input_data)
        if prediction[0] == 0:
            st.success("   R sultat : Trafic Normal")
        else:
```

```
            st.error("       R sultat : Attaque D tect e"
                )
    except Exception as e:
        st.error(f"Erreur lors de la pr diction : {e}")
```

Listing 5 – App.py - Interface Streamlit pour la prédiction

5.2 Lancement depuis le terminal

Commande pour lancer l'application Ouvrir un terminal, se placer dans le dossier contenant `App.py` et exécuter :

`streamlit run App.py`

Exemple de sortie observée dans le terminal

- Streamlit démarre et affiche un message d'accueil.
- Il fournit deux URL :
  - **Local URL** : `http://localhost:8501` (accessible depuis la même machine).
  - **Network URL** : `http://192.168.40.129:8501` (accessible depuis d'autres machines du réseau local).
- Remarque : si le terminal affiche *"You can now view your Streamlit app in your browser."* l'application est opérationnelle.

Figure 19 – Terminal Streamlit — sortie au lancement

Explication utile pour l'encadrant

- Les fichiers requis dans le même dossier que `App.py` : `random_forest_model.pkl` (modèle sauvegardé).

— Si vous avez appliqué un `scaler` et/ou un `OneHotEncoder` au training, il faut également charger les artefacts correspondants (ex : `scaler.pkl`, `encoder.pkl`) et appliquer les mêmes transformations à `input_data` avant appel à `model.predict()`.
— Si vous ne faîtes pas l'encodage/normalisation identique, vous obtiendrez des erreurs du type `could not convert string to float` ou des prédictions non-sensées.

5.3 Interface graphique (captures et explications)

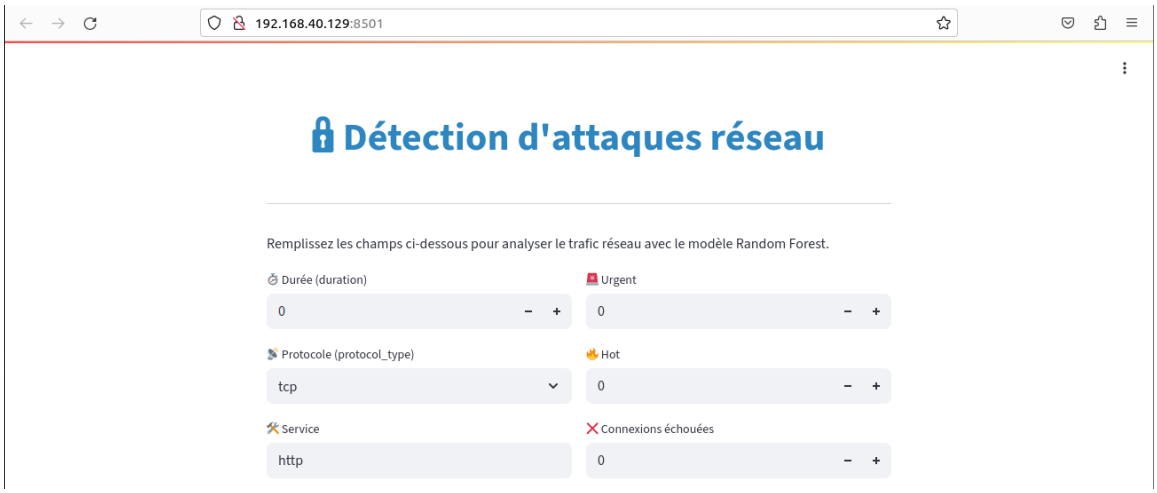


Figure 20 – Interface Streamlit - partie haute : titre, instructions et colonne de champs (extrait)

Interprétation de la figure 20 La capture montre l'en-tête de l'application et la disposition en colonnes des champs de saisie. Les widgets Streamlit (number_input, selectbox, slider) permettent d'entrer des valeurs numériques et catégorielles facilement. L'utilisateur complète ces champs puis clique sur *Lancer la prédiction*.

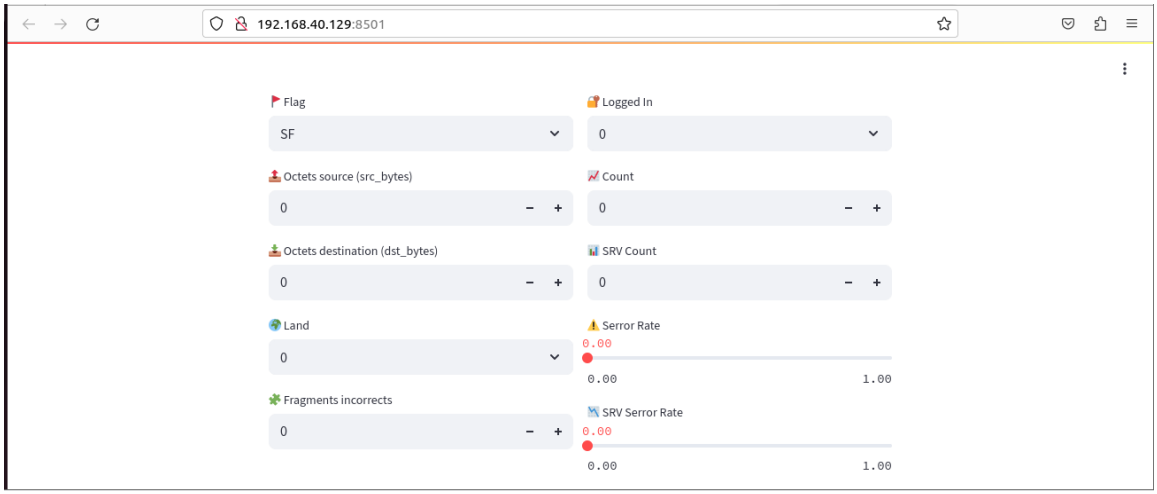


Figure 21 – Interface Streamlit - suite des champs : sliders et autres paramètres

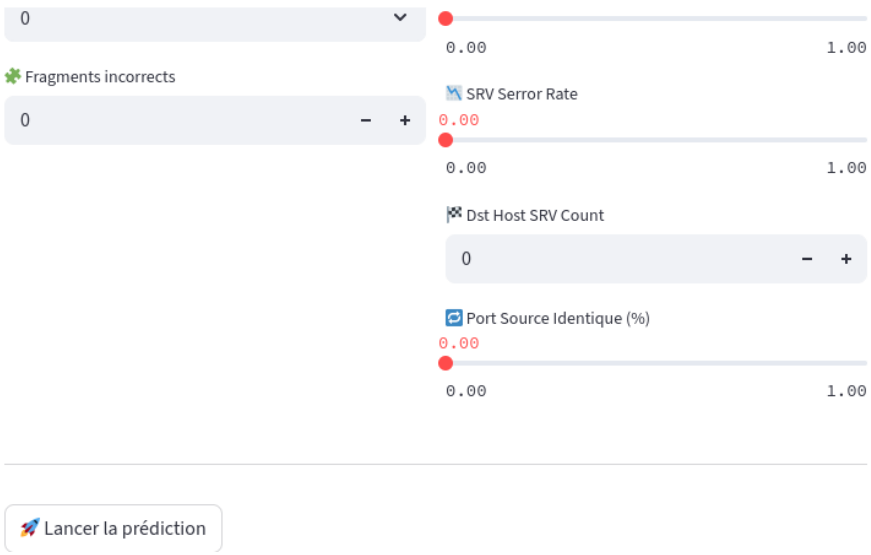


Figure 22 – Interface Streamlit - suite des champs : sliders et autres paramètres

Interprétation de la figure : Cette capture présente la partie basse du formulaire (sliders pour taux, autres compteurs). Les sliders sont pratiques pour renseigner des ratios (ex : `serror_rate`) entre 0 et 1.

Procédure de test sur l'interface (exemple concret) Pour vérifier que l'application fonctionne et tester une prédiction de type DoS/attaque, remplir les champs comme suit (valeurs représentatives) :

— `duration` = 0
— `protocol_type` = `tcp`
— `service` = `http`
— `flag` = `SF`
— `src_bytes` = 1000
— `dst_bytes` = 500
— `count` = 50
— `serror_rate` = 0.8

Puis cliquer sur `Lancer la prédiction`. Attendu : l'interface renvoie « **Résultat : Attaque Détectée** » si le modèle estime que ces caractéristiques correspondent à une attaque.

Remarques techniques importantes

— Si le modèle a été entraîné après encodage (OneHot ou LabelEncoder) et normalisation, il est impératif d'appliquer les mêmes transformations aux champs avant la prédiction. Sinon Streamlit lèvera une erreur `ValueError: could not convert string to float: 'tcp'` ou le `model.predict()` échouera.
— Pour automatiser l'encodage/standardisation dans l'application, sauvegarder et recharger (avec `joblib`) les objets `encoder` et `scaler` utilisés lors du trai-

ning, puis appliquer `encoder.transform(...)` et `scaler.transform(...)` sur `input_data`.

— En production, prévoir des contrôles d'entrée (validation) et un mécanisme de journalisation (log) des prédictions.

6 Tests prévus

Objectif : Test des attaques réelles et génération des prédictions

Tester le modèle via l'interface Streamlit avec des valeurs représentatives d'attaques (DoS, Probe, R2L, U2R), collecter les résultats, analyser les erreurs (faux positifs / faux négatifs), documenter les tests et préparer le rapport final.

— Exemple de test avec valeurs explicatives

Pour vérifier le bon fonctionnement du modèle déployé dans l'interface Streamlit, nous proposons un exemple de prédiction à partir de valeurs d'entrée simulant une attaque de type **DoS**.

Table 2 – Exemple de valeurs de test pour Streamlit

Champ	**Valeur testée**	**Description / Utilité**
`duration`	0	Durée de la connexion (en secondes). Un `0` est fréquent dans les attaques rapides DoS.
`protocol_type`	tcp	Protocole de la connexion (TCP/UDP/ICMP).
`service`	http	Service ou port applicatif utilisé (ex : HTTP, FTP, SMTP...).
`flag`	SF	Statut de la connexion TCP (SYN/FIN/-REJ). SF indique une connexion terminée normalement.
`src_bytes`	2000	Nombre d'octets envoyés par la source.
`dst_bytes`	50	Nombre d'octets reçus depuis la destination. Faible valeur côté retour pour une DoS.
`count`	50	Nombre de connexions vers le même hôte dans un laps de temps donné.
`srv_count`	50	Nombre de connexions vers le même service.
`serror_rate`	1.0	Proportion de connexions SYN ayant des erreurs.
`same_srv_rate`	1.0	Proportion de connexions au même service. Valeur haute dans un flood.

Ces valeurs sont passées dans l'interface Streamlit (Figure **??**), qui renvoie ensuite une prédiction :

— **0** : trafic normal
— **1** : attaque détectée
L'exemple ci-dessus devrait produire une **détection positive** (attaque).

6.1 Améliorations envisagées

Plusieurs pistes d'amélioration ont été identifiées :
— **Intégration de données réelles** : tester le modèle sur des logs réseau issus d'un environnement de production pour renforcer sa robustesse.
— **Comparaison avec d'autres modèles** : évaluer des algorithmes alternatifs comme *SVM*, *Logistic Regression*, *XGBoost*, ou encore des approches de *Deep Learning*, afin de justifier le choix final.
— **Déploiement avancé** : envisager l'intégration du modèle dans un environnement cloud ou au sein d'un SOC (Security Operations Center) pour une utilisation en conditions réelles.

7 Conclusion

Ce stage au sein du Groupe MANAGEM a constitué une expérience à la fois enrichissante sur le plan technique et formatrice sur le plan professionnel. L'objectif principal était de concevoir et mettre en œuvre un système de détection d'attaques réseau basé sur des techniques de Machine Learning, en exploitant des données publiques issues du dataset NSL-KDD.

La réalisation de ce projet a nécessité l'acquisition et la mise en pratique de compétences variées : analyse et prétraitement de données, compréhension et implémentation d'algorithmes de classification, évaluation et interprétation des résultats, ainsi que développement d'une interface de déploiement accessible via Streamlit. Le choix du modèle *Random Forest* a permis d'obtenir d'excellentes performances en termes de précision et de rappel, démontrant la pertinence des forêts aléatoires pour ce type de problématique. Toutefois, des comparaisons avec d'autres modèles (SVM, Régression Logistique) ont montré que le choix du modèle doit toujours être justifié par un compromis entre performance, temps de calcul et robustesse.

Ce travail a également mis en lumière certaines limites, telles que la dépendance du modèle aux données d'entraînement, le risque de sur-apprentissage, ou encore le manque de diversité dans le dataset utilisé. Ces constats ouvrent la voie à plusieurs pistes d'amélioration, notamment :

— Enrichir le dataset avec des flux réseau réels collectés au sein de l'entreprise.
— Tester des modèles plus récents comme le Gradient Boosting ou des approches de Deep Learning.
— Intégrer le modèle dans une architecture de surveillance réseau en temps réel.

Au-delà de l'aspect purement technique, ce stage a été l'occasion de développer des compétences transversales : travail collaboratif avec les équipes de cybersécurité, gestion de projet, adaptation aux contraintes opérationnelles, et communication des résultats de manière claire et synthétique. Il a également renforcé ma capacité à proposer des solutions concrètes répondant à des enjeux réels de sécurité informatique.

En conclusion, cette expérience m'a permis de mettre en pratique mes connaissances académiques dans un environnement professionnel exigeant, tout en consolidant mes compétences en Machine Learning appliqué à la cybersécurité. Les résultats obtenus constituent une base solide pour de futurs développements, et ouvrent la voie à la mise en place d'un système de détection d'attaques performant et adaptable aux évolutions constantes des menaces.

Annexes

A. Extraits de code

```
# Exemple : sauvegarde du pipeline
joblib.dump(pipeline, "rf_pipeline.pkl")
```

B. Valeurs de test (exemples)

— **Test normal (ex.)** : duration=0, protocol=tcp, service=http, flag=SF, src=181, dst=5450, count=2, srv$_c ount = 2, serror_r ate = 0.0$**Test attaque (ex.)** : $duration = 0, protocol = icmp, service = eco_i, flag = REJ, src = 0, dst = 0, count = 200, srv_c ount = 200, serror_r ate = 1.0$

Informations personnelles :
Nom : EL GHAIT Imane
Formation : École nationale des sciences appliquées, Université Sultan Moulay Slimane
Filière : Ingenierie des Réseaux Intelligents & Cybersécurité
Sujet : Machine Learning pour la cybersécurité : Détection d'attaques réseau à partir de données publiques
Encadrants : Mr CHAHIR Hafed (Managem), Mr MALEH Yassine (ENSA)
Période : Juillet – Août 2025