Projet protection - Rapport

Étudiants :

Zoé Marquis,

Charlotte Kruzic,

Daniil Kudriashov,

Ekaterina Zaitceva

Date de rendu : 21/11/2024

GitHub : https://github.com/zoemarquis/projet_protection

Sommaire :

Présentation du projet **2**
Données physiques **2**
Analyse exploratoire 3
Description des données 3
Nettoyage des Données 3
Modèles 4
CNN1D 4
Préparation des données pour les modèles KNN, CART, Random Forest, XGBoost et MLP 6
KNN 8
CART 9
Random Forest 10
XGBoost 11
MLP 12
Conclusion 12
Données réseaux **14**
Analyse exploratoire 14
Présentation des jeux de données 14
Préparation des données 14
Réduction de dimension 15
Préparation pour l'analyse PCA et les modèles 15
Données finales 16
Préparation des données pour les modèles 16
Modèles 17
Préparation des modèles 17
CART 17
Random Forest 17
XGBoost 17
MLP 17
Performances pour la classification binaire (Label_n) 17
Conclusion 18
Performances pour la classification multi-classe (Label) 18
DoS 18
Physical fault 19
MITM 19
Anomaly 19
Conclusion **20**
Contributions personnelles / Partage des tâches **21**
Zoé Marquis 21
Charlotte Kruzic 22
Daniil Kudriashov 23
Ekaterina Zaitceva 24

Présentation du projet

Ce projet a pour objectif de développer des modèles de détection d'intrusions dans un système cyber-physique simulant une distribution d'eau, basé sur le banc d'essai décrit dans l'article *A Hardware-in-the-Loop Water Distribution Testbed Dataset for Cyber-Physical Security Testing*.

Ce système associe des données physiques et des données réseau, collectées dans un environnement combinant des composants réels et simulés.

Les données sont étiquetées selon deux niveaux. Le premier est une classification binaire entre trafic "normal" et "attaque", et le second détaille le type d'attaque. Les différents types d'attaques présents sont : le *DoS* (Denial of Service), une surcharge intentionnelle d'un service par un afflux massif de requêtes ; le *MITM* (Man-in-the-Middle), une interception et modification des communications entre deux entités ; le *scan*, une exploration des ports et identification de vulnérabilités potentielles ; les défauts physiques, des anomalies physiques non causées par une cyberattaque, généralement indétectables via des données réseau ; et les anomalies diverses, qui regroupent les attaques autres que celles présentées ci-dessus.

Nous avons d'abord préparé les données, puis les avons exploitées, notamment en réduisant leurs dimensions lorsque cela s'avérait pertinent pour optimiser l'efficacité des modèles. Ensuite, nous avons testé une variété de modèles afin d'explorer différentes approches. En parallèle, nous avons créé des visualisations et effectué des analyses approfondies pour mieux comprendre et interpréter les résultats ainsi que l'utilisation des ressources.

Données physiques

Analyse exploratoire

Description des données

Le jeu de données comprend cinq fichiers distincts :

- 1 fichier contenant exclusivement du trafic normal
- 4 fichiers contenant différents types d'attaques : *DoS*, *Scan*, *Physical Fault*, et *MITM* mais aussi des données normales

Chaque fichier comporte 43 colonnes, dont :

- `Label_n` : indique si une observation est une attaque (1) ou normale (0),
- `Label` : précise la classe d'attaque ou indique "normal". Une vérification a confirmé la cohérence entre ces deux colonnes.
- `Time` : indique le temps dans cette série temporelle, où les mesures sont généralement espacées d'une seconde. Une anomalie d'espacement a été détectée mais n'a pas impacté nos analyses.
- Des colonnes `Tank` et `Flow_sensor` (entiers), ainsi que `Pump` et `Valve` (booléens).

Nous avons également étudié la possibilité que `Pump` et `Valve` soient exclusifs (comme dans un encodage one-hot), mais ce n'est pas le cas.

Distribution temporelle des attaques :

- Les attaques de type *scan* sont rares et brèves,
- Tandis que les attaques *MITM* et *DoS* tendent à s'étendre sur des durées plus longues.

Nettoyage des Données

Voici les principales étapes de nettoyage effectuées :

- Renommage des colonnes et labels pour corriger des incohérences, comme `Lable_n` remplacé par `Label_n` et "nomal" corrigé en "normal".
- Suppression des colonnes contenant une seule valeur, jugées non informatives. Pour plus de détail : voir le notebook `a_preparation_phy.ipynb`
- Aucune donnée manquante n'a été détectée, aucune action spécifique n'était nécessaire.
- Analyse et transformation des colonnes `Flow_sensor_1` et `Flow_sensor_2` : Une analyse plus détaillée des colonnes a révélé des caractéristiques spécifiques dans la distribution de leurs valeurs. Ces observations ont conduit à des transformations adaptées, visant à simplifier leur utilisation tout en préservant leur pertinence sémantique.
    - Transformation de `Flow_sensor_2`:
        - Les valeurs de cette colonne se répartissent en deux groupes distincts: 0 et 4000.

        - Cette distribution binaire indique qu'il est plus pertinent de traiter cette colonne comme une variable booléenne plutôt que comme une variable numérique continue. Ainsi :
            - 0 est transformé en `False`.
            - 4000 est transformé en `True`.
        - Cette transformation réduit le bruit inutile lié à l'échelle numérique tout en conservant l'information clé sur la présence ou l'absence d'un signal.
    - Transformation de `Flow_sensor_1`:
        - Les valeurs observées sont :
            - 0 et 4000 dans tous les datasets.
            - Une valeur supplémentaire, 100, présente uniquement dans df_phy_1.
        - Ces valeurs discrètes ne varient pas de manière continue, contrairement à d'autres colonnes comme `Flow_sensor_4`.
        - Pour refléter correctement cette nature non continue et faciliter l'encodage dans les modèles, cette colonne a été transformée en une variable catégorielle.

Modèles

CNN1D

Les données physiques que nous analysons sont issues de séries temporelles, où chaque observation est prise à intervalles réguliers d'une seconde. Ce type de données est idéalement adapté aux réseaux de neurones convolutifs (CNN) 1D, qui sont spécifiquement conçus pour traiter des séquences d'observations structurées dans le temps.

Les séries temporelles, par définition, sont constituées de données collectées successivement dans le temps, avec des dépendances souvent complexes entre les valeurs successives. Tout comme les images dans un CNN 2D, ces séries présentent des motifs et des tendances locales qu'il est crucial de capturer pour comprendre et prédire les comportements futurs. Un CNN 1D est particulièrement efficace pour cette tâche, car il peut extraire des caractéristiques locales à différentes échelles temporelles en glissant des filtres de convolution le long de la séquence de données.

L'efficacité des CNN 1D dans la détection de motifs locaux et leur capacité à capter des informations temporelles complexes les rend particulièrement adaptés aux tâches telles que la classification de séries temporelles ou la détection d'anomalies, comme c'est le cas dans notre analyse des données physiques. Ce type d'architecture permet de comprendre non seulement les tendances générales des séries temporelles, mais aussi d'identifier les événements spécifiques qui ont un impact significatif sur le comportement global des données, comme les attaques ou autres anomalies dans les séries.

Préparation des données

- Les différentes sources de données ont été fusionnées pour former une série temporelle complète tout en respectant l'ordre temporel logique. La colonne

timestamp a été supprimée, car l'information temporelle est déjà implicite dans la position des données au sein de la séquence.

- Des fenêtres glissantes de taille fixe (10 secondes) ont été générées indépendamment pour chaque dataset, chaque fenêtre devenant une observation pour l'entraînement. Après avoir testé plusieurs tailles, une fenêtre de 10 secondes a montré des résultats satisfaisants.
- Les données ont ensuite été divisées en :
    - Variables descriptives (X), représentant les caractéristiques.
    - Étiquettes (y), comme `Label` (classe) ou `Label_n` (attaque ou normal)
- Une colonne est ajoutée pour identifier l'origine des données (dataset).
- Les données ont été normalisées à l'aide d'un `MinMaxScaler` pour garantir que toutes les caractéristiques contribuent de manière équilibrée au processus d'apprentissage.
- Encodage des données :
    - *One-hot encoding* pour les colonnes catégorielles.
    - *Label encoding* pour les classes, avec un encodage commun pour assurer la cohérence entre les modèles.
- Division en ensembles d'entraînement et de test.

Entraînement des modèles

Deux approches principales ont été adoptées pour l'entraînement des modèles :

1. Prédiction binaire : Modèle entraîné pour distinguer les états *attaque* ou *normal* (prédiction de `Label_n`).
2. Classification multiclasses : Un modèle unique est conçu pour prédire plusieurs classes simultanément (`Label`).

Pour la classification binaire :

- Architecture CNN 1D simple et efficace :
    - Deux couches de convolution suivies de *max-pooling*
    - Une couche *flatten*.
    - Deux couches *dense*, activées respectivement par *ReLU* et *Sigmoid* (pour la prédiction binaire).
    - Utilisation de l'optimiseur Adam et de la fonction de perte *binary_crossentropy*.

Pour la classification multiclasses :

- Une architecture similaire avec une couche *dense* finale utilisant une activation *Softmax*.
- Fonction de perte *sparse_categorical_crossentropy* avec le même optimiseur Adam.

Remarque : Des architectures plus complexes, intégrant d'autres couches, ont été testées, mais une structure simple s'est avérée plus performante pour ce cas d'usage.

Optimisation commune : Dans les deux approches, *early stopping* a été utilisé pour prévenir le surapprentissage et améliorer l'efficacité de l'entraînement.

Analyse de sa performance

Pour la classification binaire :

Les performances du modèle pour la classification binaire sont excellentes. Toutes les métriques équilibrées (balanced) sont supérieures à 0.94, ce qui montre un très bon ajustement du modèle. Pour les classes déséquilibrées (imbalanced), bien que les résultats soient légèrement inférieurs, ils restent néanmoins très élevés, avec des scores au-dessus de 0.93, ce qui est remarquable.

Pour la classification multiclasses :

Les résultats des métriques de classification varient selon les types d'attaques. Pour le ***scan***, les performances sont nettement moins bonnes en raison du manque de valeurs suffisantes, ce qui limite la précision du modèle dans ce domaine. En revanche, pour les données ***Physical Fault***, le modèle se distingue en obtenant la meilleure précision et le meilleur taux de vrais négatifs (*TNR*) parmi tous les modèles associés à ces données. Concernant la détection des attaques ***MITM***, le modèle affiche des résultats remarquables avec le meilleur rappel (*Recall*) et la meilleure *Balanced Accuracy*, ainsi que les deuxièmes meilleurs scores en *accuracy*, *F1-score* et *MCC*. Il se place juste derrière le modèle *Random Forest*, qui est particulièrement performant pour ce type d'attaque. Enfin, le modèle présente des performances moyennes pour la détection des attaques par ***DoS***.

Temps d'entraînement et ressources consommées :

En termes de ressources, le modèle binaire nécessite un temps d'entraînement et de prédiction significativement plus long par rapport aux autres modèles. Le temps d'entraînement est environ 20 fois plus long, et le temps de prédiction est environ 2 fois plus long, bien que ces valeurs restent faibles en absolu. En ce qui concerne la mémoire, le modèle utilise environ 10 fois plus de mémoire pendant l'entraînement, principalement en raison de l'espace nécessaire pour toutes les fenêtres intrinsèques aux modèles. Pour la prédiction, la consommation de mémoire est plus élevée que pour les autres modèles, mais reste tout de même dans des valeurs acceptables.

En conclusion, l'utilisation du modèle CNN1D s'est avérée efficace pour extraire des *features* pertinentes à partir des données, offrant ainsi de bonnes performances globales.

Préparation des données pour les modèles KNN, CART, Random Forest, XGBoost et MLP

Pour l'ensemble des modèles classiques (*KNN*, *CART*, *Random Forest*, *XGBoost* et *MLP*), une approche commune de préparation des données a été adoptée afin de garantir une comparaison équitable des performances.

Nettoyage et sélection des caractéristiques

Les données ont d'abord été nettoyées en :

- Supprimant la colonne temporelle ("*Time*") qui n'apporte pas d'information pertinente pour la classification
- Séparant les caractéristiques (*features*) et les étiquettes (*labels*)

- Conservant les deux types d'étiquettes : binaire (`Label_n` pour normal/attaque) et multiclasse (`Label` pour le type spécifique d'attaque)

Prétraitement des données

Plusieurs étapes de prétraitement ont été appliquées :

1. Concaténation des datasets : Avant toute transformation, les différents datasets (`df_phy_1`, `df_phy_2`, `df_phy_3`, `df_phy_4` et `df_phy_norm`) ont été combinés en un seul afin de garantir une cohérence des colonnes et d'uniformiser les transformations appliquées.
2. Standardisation des caractéristiques numériques : Application de `StandardScaler` pour normaliser toutes les variables numériques Cette étape est particulièrement importante pour les algorithmes sensibles à l'échelle comme *KNN* et *MLP*
3. Encodage des variables catégorielles : Les colonnes catégorielles ont été transformées en variables numériques via one-hot encoding. Cette méthode garantit une représentation claire des catégories et permet une compatibilité avec les modèles nécessitant des variables numériques en entrée.

Bien que certains algorithmes, comme *Random Forest*, puissent gérer directement les variables catégorielles, l'encodage one-hot a été appliqué uniformément pour assurer une comparaison équitable entre les modèles.

Aucune transformation one-hot encoding n'a été réalisée sur les colonnes de type booléen. Les données booléennes (*True*/*False*) étant déjà binaires, elles peuvent être interprétées directement par tous les algorithmes sans nécessiter d'encodage supplémentaire, ce qui évite un traitement inutile.

Division des données

Les données préparées ont ensuite été divisées en :

- Ensemble d'entraînement (80% des données)
- Ensemble de test (20% des données)
- Cette division a été réalisée avec `train_test_split` en utilisant `random_state=42` pour garantir la reproductibilité des résultats.

Préparation pour la détection des différentes classes

Pour assurer une cohérence dans l'encodage des classes (colonne `Label`) entre les différents modèles, les étapes suivantes ont été mises en place :

- Utilisation de `LabelEncoder` : Transformation des classes (`Label`) en valeurs numériques pour une meilleure compatibilité avec les algorithmes d'apprentissage machine.
- Construction d'un dictionnaire de *mapping* : Création d'un dictionnaire permettant de maintenir la traçabilité des classes, en garantissant une correspondance stable et cohérente des étiquettes numériques à leurs classes d'origine.

Cette préparation commune des données permet d'assurer que les différences de performance observées entre les modèles sont dues à leurs caractéristiques intrinsèques plutôt qu'à des variations dans le prétraitement des données.

KNN

Nous avons exploré deux versions du modèle : une utilisant directement les caractéristiques d'origine, et une autre exploitant les données réduites par PCA. Cette double approche nous a permis d'évaluer l'impact de la réduction de dimensionnalité sur les performances du modèle. L'utilisation de la PCA visait non seulement à réduire la dimensionnalité des données tout en préservant l'information pertinente, mais aussi à potentiellement améliorer les performances en éliminant le bruit et à réduire les temps de calcul des distances entre points.

Entraînement des modèles

Nous avons d'abord abordé le problème sous l'angle d'une classification binaire, en distinguant simplement les états normaux des attaques (pour prédire `Label_n` comme pour prédire `Label`). Pour cette tâche, un modèle *KNN* avec k=5 voisins a été entraîné sur l'ensemble des données, en utilisant la distance euclidienne comme métrique de similarité.

Pour la classification des différents types d'attaque, nous avons expérimenté deux approches distinctes :

- La première consistait en une classification multiclasse directe, où un seul modèle *KNN* était utilisé pour classifier toutes les classes simultanément. Cette approche s'est cependant révélée moins performante que l'approche suivante.
- La seconde approche transforme le problème multiclasse en plusieurs classifications binaires de type "une classe contre toutes". Cette stratégie implique l'entraînement d'un modèle *KNN* distinct pour chaque type d'attaque. Bien que cette approche nécessite l'entraînement de plusieurs modèles, elle reste parfaitement viable dans notre contexte, les données physiques étant relativement légères et l'entraînement des modèles *KNN* rapide.

Face au déséquilibre important des classes, particulièrement prononcé pour les attaques de type *scan*, nous avons tenté d'appliquer des techniques d'***oversampling***. Cependant, cette approche n'a pas permis d'améliorer significativement la détection des *scans*, suggérant que la difficulté ne provient pas uniquement du déséquilibre des classes, mais potentiellement aussi de la nature même de ces attaques et de leur représentation dans les données physiques. On pourra faire ce même constat pour tous les modèles suivants.

Analyse de sa performance

Pour la classification binaire (dans le cas de la prédiction de `Label_n`), le modèle *KNN* classique démontre d'excellentes performances avec des métriques équilibrées élevées : une précision de 0.956, un rappel de 0.973 et une *balanced accuracy* de 0.981. La version avec la *PCA* montre des performances légèrement différentes, toutes un peu plus faibles que celles du *KNN* sans la *PCA*.

Les performances de classification varient selon le type d'attaque, mais elles restent globalement très bonnes pour la majorité des classes. Cependant, certaines observations méritent d'être soulignées :

- *Scan* : Les résultats pour cette classe ne sont pas concluants, principalement en raison de sa sous-représentation dans les données. Malgré des tentatives d'oversampling pour rééquilibrer cette classe, aucune amélioration significative n'a été observée.
- *DoS* : Le modèle atteint ses meilleures performances sur cette classe, avec des scores élevés sur toutes les métriques, y compris le rappel, l'*accuracy*, le *F1-score*, la *balanced accuracy*, et le *Matthews Correlation Coefficient* (*MCC*).

- *Physical Fault* : Les résultats sont excellents, le modèle étant le meilleur sur toutes les métriques pour cette classe.
- *MITM* : Les performances pour cette classe sont modérées, avec des scores moyens comparés aux autres classes.

La comparaison avec les résultats de l'article est très proche de notre modèle.

Analyse des performances en terme de temps d'entraînement, prédiction et mémoire :

- Entraînement : Avec et sans *PCA*, l'entraînement est très rapide, particulièrement en comparaison avec *CNN1D*, qui est nettement plus exigeant en temps. Les modèles nécessitent peu de mémoire pour l'entraînement, environ 4 à 5 fois moins que *CNN1D* pour le modèle sans PCA et environ 7 fois moins pour le modèle avec PCA.
- Prédiction : *KNN* sans PCA est particulièrement gourmand en mémoire, nécessitant environ 60 fois plus que *CNN1D*, qui reste le modèle le plus exigeant après *KNN*. En revanche, *KNN* avec PCA bénéficie d'une réduction significative de l'utilisation mémoire, avec des besoins presque 100 fois inférieurs à ceux de *KNN* sans PCA, grâce à la réduction dimensionnelle. *KNN* sans PCA est légèrement plus long que *CNN1D* en temps de prédiction mais *KNN* avec PCA est deux fois plus rapide tout en nécessitant des ressources beaucoup plus modestes. Cette version avec PCA réduit l'empreinte mémoire pour la classification des types d'attaques. On note tout de même que ce pire temps de prédiction reste raisonnable, sous la seconde.

Ces valeurs sont rapportées pour la détection de la colonne `Label_n`, mais les tendances observées (en termes d'échelles, de classification des algorithmes les plus rapides ou les plus lents, et des besoins en mémoire) restent globalement similaires pour les autres modèles.

CART

Pour tous les modèles restants, nous avons également adopté une double stratégie de classification : une approche binaire pour la détection générale d'attaques (`Label_n`) et une approche de type "une contre toutes" pour l'identification spécifique des types d'attaques (les différents `Label`s).

Entraînement des modèles

Pour la classification binaire attaque/normal, nous avons utilisé un `DecisionTreeClassifier` de scikit-learn sans limitation de profondeur maximale, permettant à l'arbre de se développer naturellement jusqu'à atteindre une séparation optimale des classes. Cette approche a été choisie après avoir constaté qu'une limitation de la profondeur n'améliorait pas significativement les performances tout en risquant de perdre des patterns importants.

L'approche "une contre toutes" implique l'entraînement d'un arbre de décision distinct pour chaque type d'attaque, transformant ainsi le problème multiclasse en une série de classifications binaires.

Pour chaque modèle, nous avons utilisé les critères de séparation de Gini pour évaluer la qualité des divisions, ce choix étant motivé par sa robustesse et sa rapidité de calcul. L'algorithme utilise naturellement l'importance des caractéristiques pour construire l'arbre, ce qui nous permet également d'identifier les variables les plus déterminantes pour la détection de chaque type d'attaque.

Face au problème des attaques de type *scan* sous-représentées, l'approche par arbres de décision offre l'avantage de pouvoir ajuster les poids des classes. Cependant, même avec cet ajustement, la détection des *scans* reste un défi, suggérant que la difficulté réside davantage dans la nature des caractéristiques disponibles que dans la méthode de classification utilisée.

Analyse de sa performance

Les performances du modèle sont globalement moyennes : il n'est ni parmi les meilleurs ni les pires pour la détection de `Label_n`. Sa précision pour *DoS* est élevée (0.954), mais reste parmi les moins bonnes. Il présente un *TNR* moins bon pour *MITM* et est moins performant pour *Physical Fault*, se classant avant-dernier sur toutes les métriques, bien que restant au-dessus de 0.9. Pour *scan*, les résultats sont similaires à ceux mentionnés précédemment, avec des valeurs inférieures aux autres mais dans la même échelle.

En termes de temps d'entraînement, il est très rapide, et les temps de prédiction sont également parmi les plus rapides. La mémoire utilisée est très faible, tant pendant l'entraînement que la prédiction, ce qui rend ce modèle particulièrement efficace en termes de ressources, avec des besoins négligeables visibles à peine dans les visualisations.

Bien que les performances globales soient légèrement inférieures à celles des autres algorithmes, l'excellent compromis entre performances et ressources computationnelles fait de *CART* une option très intéressante pour les systèmes nécessitant une détection d'attaques rapide et économe en ressources.

Random Forest

Cet algorithme est particulièrement bien adapté à notre problématique de détection d'attaques car il permet de gérer naturellement les relations non linéaires entre les variables et offre une robustesse accrue face au bruit dans les données.

Entraînement des modèles

Pour la classification binaire attaque/normal, nous avons implémenté un `RandomForestClassifier` avec 100 arbres, un choix qui offre un bon compromis entre performance et temps de calcul. Chaque arbre est construit en utilisant un sous-ensemble aléatoire des données (*bootstrap sampling*) et un sous-ensemble des caractéristiques à chaque division, ce qui permet d'obtenir une diversité maximale dans l'ensemble.

La gestion du déséquilibre des classes, a été adressée de deux manières : premièrement via l'utilisation de l'approche "une contre toutes" qui permet de mieux gérer ce déséquilibre pour chaque type d'attaque individuellement, et deuxièmement en exploitant la capacité naturelle de *Random Forest* à gérer les classes déséquilibrées grâce au *bootstrap sampling*.

Les hyperparamètres ont été maintenus constants pour tous les modèles (nombre d'arbres, profondeur maximale non limitée) après avoir constaté que leur ajustement n'apportent pas d'amélioration significative des performances. Cette stabilité des performances avec les paramètres par défaut est une caractéristique appréciable de *Random Forest*, qui simplifie son déploiement en production.

Analyse de sa performance

Pour `Label_n`, le modèle obtient la meilleure précision (0.974) et *TNR* (0.994), et le deuxième meilleur *MCC* (0.954), avec des résultats moyens pour les autres métriques. Pour *DoS*, il atteint la meilleure précision et *TNR* (1), mais ses performances sont moyennes pour les autres métriques. Pour *MITM*, il excelle avec la meilleure précision et *TNR*, et se classe

deuxième sur toutes les autres métriques, avec des valeurs très élevées. Il est moyen pour *Physical Fault* et présente les mêmes résultats que les autres modèles pour *scan*.

En termes de mémoire, il est très léger en entraînement, nécessitant environ trois fois moins de mémoire que *KNN* et deux fois moins que *MLP*. Il est également parmi les moins gourmands pour la prédiction. Le temps d'entraînement est relativement élevé, deuxième après *CNN1D* et comparable à *XGBoost*, avec environ 1,5 seconde d'entraînement et moins de 0,2 seconde pour la prédiction, des temps constants pour `Label_n` et les différentes attaques.

XGBoost

Contrairement à *Random Forest* qui crée des arbres indépendants, *XGBoost* construit ses arbres de manière séquentielle, chaque nouvel arbre cherchant à corriger les erreurs des arbres précédents. Cette approche est particulièrement intéressante pour notre problématique de détection d'attaques car elle permet potentiellement de capturer des *patterns* plus subtils dans les données physiques.

Entraînement des modèles

Pour la classification binaire attaque/normal, nous avons configuré *XGBoost* avec 100 estimateurs (arbres), un taux d'apprentissage de 0.1 et une profondeur maximale de 5 pour chaque arbre. Ces paramètres ont été choisis pour offrir un bon compromis entre la capacité du modèle à capturer des *patterns* complexes et le risque de surapprentissage. L'utilisation de la fonction de perte "*logloss*" est particulièrement adaptée à notre tâche de classification binaire.

Pour la classification des différentes classes d'attaque, nous avons maintenu l'approche "une contre toutes" qui s'est avérée efficace avec les autres modèles.

Le déséquilibre des classes a été géré grâce aux paramètres de pondération intégrés à XGBoost. L'algorithme permet en effet d'attribuer automatiquement des poids différents aux classes en fonction de leur fréquence, ce qui aide à compenser le déséquilibre naturel des données.

Pour l'entraînement, nous avons utilisé l'*early stopping* avec une patience de 10 itérations, surveillant la progression sur un ensemble de validation. Cette approche permet d'éviter le surapprentissage tout en garantissant que le modèle atteint ses performances optimales.

Analyse de sa performance

Pour `Label_n`, le modèle est deuxième meilleur en précision (0.957) et TNR (0.993), mais moins bon en rappel (0.728) et balanced accuracy (0.860), et deuxième moins bon après *MLP* en *MCC*. Pour *DoS*, les performances sont moyennes, à l'exception du *TNR* qui est parfait. Pour *MITM*, il affiche un très bon *TNR*, mais les autres métriques sont moyennes (entre 0.8 et 0.9). Pour *Physical Fault*, le modèle est plutôt très bon, se classant deuxième meilleur sur la plupart des métriques. Pour *scan*, les résultats sont similaires à ceux des autres modèles.

Le modèle a le temps d'entraînement le plus long de tous les modèles (sauf CNN 1D), mais reste plus rapide en prédiction que *KNN* et *KNN PCA* (0.07 secondes). En termes de mémoire, il est très léger, nécessitant moins de mémoire en entraînement et prédiction que les autres modèles.

MLP

Le Perceptron Multicouches (*MLP*) représente une approche de réseaux de neurones plus classique et plus simple que le CNN1D, tout en conservant la capacité d'apprentissage de relations non linéaires complexes dans les données. Pour notre problématique de détection d'attaques dans les données physiques, le MLP offre l'avantage de pouvoir traiter directement les caractéristiques extraites sans nécessiter de structure temporelle particulière, contrairement au CNN1D.

Entraînement des modèles

Pour la classification binaire attaque/normal, nous avons implémenté une architecture *MLP* composée de deux couches cachées (100 et 50 neurones respectivement) avec activation *ReLU*. Cette structure a été choisie après expérimentation comme offrant le meilleur compromis entre capacité de modélisation et risque de surapprentissage. L'optimiseur Adam a été utilisé avec un taux d'apprentissage de 0.001 et la fonction de perte *binary_crossentropy*.

Pour prévenir le surapprentissage, nous avons mis en place un *early stopping* avec une patience de 10 époques, surveillant la perte sur un ensemble de validation. Cette approche permet d'arrêter l'entraînement lorsque les performances cessent de s'améliorer, évitant ainsi une dégradation de la capacité de généralisation du modèle.

Le problème du déséquilibre des classes a été géré via l'ajustement des poids des classes dans la fonction de perte. Cependant, comme pour les autres modèles, la détection des *scans* reste un défi, suggérant que la difficulté réside davantage dans la nature des données que dans l'architecture du modèle.

Analyse de sa performance

Pour `Label_n`, le modèle est parmi les moins performants en rappel, TNR, balanced accuracy, et MCC, bien que les valeurs restent respectables (au-dessus de 0.777). Pour *DoS*, il est dernier pour toutes les métriques (par exemple, 0.463 en rappel, 0.614 en F1 score, et 0.643 en MCC). Pour *MITM*, les résultats sont similaires, mais avec des valeurs au-dessus de 0.796. Pour *Physical Fault*, les performances sont comparables, avec 0.626 en précision et 0.754 en F1 score. Pour *scan*, les résultats suivent la même tendance que pour les autres modèles.

temps et ressource :

Le modèle a un temps d'entraînement très rapide, similaire à *KNN* et *CART*, et un temps de prédiction rapide, équivalent à *CART*. Cependant, il utilise plus de mémoire en entraînement et prédiction que *XGBoost*, *Random Forest*, *CART* et *KNN* avec *PCA*.

Conclusion

En résumé, pour les modèles des données physiques, la détection du *scan* reste impossible en raison de la nature des données. Malgré les tentatives d'*oversampling*, aucune amélioration significative des résultats n'a été observée. Les résultats obtenus pour tous les modèles sont les suivants : précision, rappel, *F1-score* et *MCC* à 0 ; *TNR* et *accuracy* à 1 ; et *balanced accuracy* à 0.5.

Nous avons également testé *PCA* avec *KNN*, ce qui a montré une amélioration en termes de ressources, mais sans amélioration des métriques.

Pour `Label_n`, en termes de précision et de rappel, les meilleurs modèles sont *CNN1D*, *KNN*, *KNN PCA*, *CART*, *Random Forest* et *KNN* article / *RF* article.

En termes de *TPR / TNR*, *CNN1D*, *KNN*, *KNN PCA*, *CART* et *Random Forest* se distinguent, tandis que *XGBoost* et *MLP* n'atteignent pas les meilleures performances.

De même, pour TPR / FPR, les mêmes modèles ressortent parmi les meilleurs.

Concernant MITM et *DoS*, le modèle MLP se révèle moins performant. Les meilleurs résultats sont obtenus avec *Random Forest*, *CNN1D* et les différents *KNN*, que ce soit pour TPR, FPR, ou TNR. Pour *Physical Fault*, *MLP* reste le moins bon modèle en termes de précision.

Globalement, toutes les métriques sont relativement élevées. À l'exception de XGBoost et MLP dans certains cas, les résultats sont constamment supérieurs à 0.95.

En termes de ressources, CNN 1D est le modèle le plus gourmand, particulièrement en entraînement, bien qu'il soit plus correct en prédiction. Les autres modèles sont rapides en entraînement, avec *KNN* étant légèrement plus long en prédiction. *KNN*, en revanche, demande beaucoup plus de mémoire en prédiction que les autres modèles.

Remarque importante : Les données présentées ci-dessous sont celles obtenues après l'exécution de l'analyse. Bien que nous ayons fixé la *seed* et pris toutes les mesures possibles pour assurer la reproductibilité des résultats, des fluctuations peuvent survenir. Par conséquent, il est préférable de considérer ces valeurs comme un ordre d'idée général plutôt que de se concentrer sur les résultats à un millième près.

Données réseaux

Analyse exploratoire

Cette section présente les étapes d'analyse exploratoire et de préparation des données issues de réseaux industriels. L'objectif est de garantir une qualité optimale des données pour leur exploitation dans des modèles de *Machine Learning* destinés à détecter et classifier différents types d'attaques.

Présentation des jeux de données

Les données utilisées proviennent d'un environnement industriel où les communications entre appareils sont statiques, ce qui confère un caractère déterministe au trafic réseau. Ces données sont réparties en cinq fichiers distincts : l'un contient uniquement du trafic normal, tandis que les quatre autres représentent différents types d'attaques. Chaque fichier inclut seize colonnes, dont quatorze correspondent à des caractéristiques techniques (adresses IP et MAC, ports, protocoles, taille et nombre de paquets, *flags*, requêtes et réponses Modbus, *timestamp*) et deux sont des étiquettes indiquant la nature des enregistrements.

L'analyse initiale a révélé des déséquilibres significatifs dans les données. Le fichier contenant du trafic normal est 1,5 fois plus volumineux que les fichiers représentant des attaques, ce qui reflète bien la réalité d'un réseau où les attaques sont rares. Parmi les types d'attaques, les *DoS* sont sur-représentés et se distinguent nettement par leurs caractéristiques des autres types d'attaques. En revanche, les *scans* restent rares, et les anomalies physiques ne se distinguent pas du trafic normal dans les caractéristiques réseau, car elles ne modifient pas les communications.

Préparation des données

Harmonisation et traitement initial

La première étape a consisté à uniformiser les jeux de données en corrigeant les noms, les types et l'ordre des colonnes. Une vérification a confirmé la cohérence entre les colonnes signalant une attaque et celles détaillant les types d'attaques.

Pour ne pas biaiser les modèles, les valeurs manquantes ont été traitées de la manière suivante : les valeurs catégorielles ou textuelles manquantes ont été remplacées par une catégorie "inconnue", tandis que les valeurs numériques ont été imputées par leur médiane. Cette approche garantit que l'information potentiellement utile contenue dans les valeurs manquantes est préservée.

Traitement des doublons

Une analyse approfondie des données d'attaques a révélé la présence de nombreuses lignes dupliquées, principalement associées aux attaques *DoS*. Pour éviter que ces doublons biaisent les résultats des modèles, une colonne indicatrice a été ajoutée, permettant de repérer ces enregistrements avant de les supprimer.

Transformation des caractéristiques réseau

Les valeurs des ports, bien que numériques, n'ont pas de sens en tant que données continues. Pour simplifier leur traitement, les ports ont été regroupés en trois catégories :

- Ports système (0-1023),
- Ports utilisateurs (1024-49151),
- Ports dynamiques ou privés (49152-65535).

source : Service Name and Transport Protocol Port Number Registry

Une tentative d'ajout d'une colonne signalant si un port était connu, sur la base des données normales, n'a pas amélioré les résultats des modèles et a donc été abandonnée.

Les *flags*, qui codent une dizaine d'états possibles, ont été convertis en valeurs catégorielles, puis en variables binaires grâce à un *one-hot encoding*, ce qui facilite leur utilisation dans les modèles.

Réduction de dimension

Les matrices de corrélation calculées pour les différents fichiers de données ont révélé des corrélations significatives entre plusieurs caractéristiques. Par exemple, des liens évidents ont été observés entre les adresses IP sources et destinations, les ports sources et de destination, le nombre de paquets envoyés et reçus, ainsi que les réponses Modbus. De plus, des corrélations existent entre les protocoles de communication et les flags. Ces résultats mettent en évidence une redondance importante dans l'information contenue dans les données, ce qui pourrait introduire des biais ou ralentir l'apprentissage des modèles.

Pour réduire cette redondance et diminuer la quantité de données à traiter par les modèles, une réduction de dimension a été effectuée. Cette démarche permet non seulement d'éliminer les variables redondantes, mais aussi de simplifier la structure des données tout en conservant les informations les plus significatives. La méthode choisie pour cette réduction est l'Analyse en Composantes Principales (*PCA*). Cette méthode a été préférée à des alternatives comme *t-SNE* ou *UMAP* en raison de son aptitude à conserver la variance globale des données et de sa rapidité d'exécution, particulièrement adaptée aux volumes importants de données réseau que nous avons.

La *PCA* a été appliquée de 3 façons différentes : séparément sur chaque fichier, sur une concaténée des cinq fichiers, et sur une version concaténée des quatre fichiers d'attaques. L'objectif était de conserver au moins 90 % de la variance cumulée.

Les résultats montrent qu'il est possible de réduire les 33 dimensions initiales à seulement 6 dimensions, tout en préservant au moins 90 % de la variance dans l'ensemble des fichiers, à l'exception du fichier d'attaque 3 qui atteint cette proportion avec seulement 4 dimensions.

Préparation pour l'analyse PCA et les modèles

Transformation de la variable temporelle

La colonne `Time`, au format *datetime*, a été supprimée car elle n'apportait pas d'information pertinente pour la *PCA*. Nous avions initialement tenté de dériver des colonnes telles que l'année, le mois, le jour, l'heure, la minute, ou encore le delta de temps entre les enregistrements. Cependant, ces transformations ont conduit à une domination de la dimension temporelle dans l'analyse, sans pour autant améliorer la détection des attaques.

Normalisation et encodage

Pour garantir une contribution équitable des différentes colonnes dans les analyses, toutes les variables numériques ont été normalisées, et les colonnes catégorielles ont été traitées selon leur nature :

- Celles avec peu de catégories, comme les *flags*, ont été encodées en *One-Hot*.
- Les colonnes présentant un grand nombre de catégories, telles que les réponses Modbus, ont été regroupées. Seules les catégories représentant au moins 0,1 % des données ont été conservées, les autres étant regroupées sous une catégorie "autres", avant d'être encodées en One-Hot.

- Les colonnes textuelles ou de type "object" ont été encodées de manière ordinale, puis standardisées.

Les colonnes booléennes ont été converties en valeurs numériques de type float pour garantir leur prise en compte correcte par les modèles.

Exclusion des variables non pertinentes

Les colonnes introduites pour gérer les valeurs manquantes ont été supprimées dans le jeu de données final, tout comme les étiquettes (de détection et de type d'attaque) lors de l'analyse *PCA*, pour éviter qu'elles influencent les résultats.

Données finales

Après préparation, le jeu de données final contient 33 colonnes prêtes à être utilisées pour la réduction de dimension et l'entraînement des modèles. Cette préparation a permis de structurer les données tout en intégrant des informations cruciales sur les types d'attaques, garantissant ainsi leur pertinence pour la détection de comportements malveillants.

Préparation des données pour les modèles

Parmi les trois types de regroupement de données pour la PCA, nous avons conservé la méthode qui combine les quatre jeux de données d'attaques, ce qui permet d'obtenir une variété de types d'attaques (environ 20 millions d'entrées). Ensuite, nous avons sélectionné un échantillon représentant 25 % de ces données (tout en préservant la distribution originale des différentes attaques) afin d'obtenir un jeu de données plus adapté à l'analyse par *PCA* (environ 5 millions d'entrées).

Après cette étape, le jeu de données contient environ 58 % de données normales, 24 % de *DoS*, 10 % de *MITM*, 7 % de *physical fault*, et le reste constitué des types *anomaly* et *scan*.

L'utilisation de **6 dimensions dans la PCA** nous a permis de conserver **plus de 90 %** de la variance.

À partir des résultats de la PCA, nous avons extrait un jeu de données-échantillon représentant **10 % des données** (soit 500 000 entrées) pour les modèles. Cela permet de conserver une partie significative des données tout en réduisant la taille du jeu de données afin qu'il reste gérable en termes de calcul et de mémoire.

Enfin, nous avons préparé les données pour les classifications en séparant les caractéristiques des étiquettes (X et y). Les données ont été divisées en ensembles d'entraînement et de test, avec une stratification pour préserver la distribution des classes.

Pour équilibrer les classes dans l'ensemble d'entraînement, nous avons appliqué un oversampling à l'aide de la technique **SMOTE (Synthetic Minority Oversampling Technique)**. Cela a permis de générer des exemples synthétiques pour les classes minoritaires, comme les attaques de type *MITM* ou *physical fault*, afin de rééquilibrer le jeu de données. Avant cette étape, le jeu de données était fortement déséquilibré, avec une majorité de données normales et *DoS* et peu de données représentant d'autres types d'attaques. L'utilisation de SMOTE a permis à nos modèles d'apprendre à mieux détecter les différents types d'attaques, tout en évitant d'être biaisé en faveur des classes majoritaires.

À la fin de la préparation, les jeux de données d'entraînement et de test pour les classifications binaire et multi-classes ont été sauvegardés dans une base de données *PickleShare*.

Modèles

Pour tous les modèles, nous avons appliqué les mêmes types de classification.

La première classification est de type **binaire**, permettant de distinguer les données normales des attaques (prédiction de `Label_n`).

La deuxième classification est de type **multi-classe**, permettant de classer les données en fonction du type d'attaque spécifique (prédiction de `Label`).

Préparation des modèles

KNN

Le modèle ***KNN* (K-Nearest Neighbors)** utilise les 10 voisins les plus proches pour prédire la classe d'un échantillon. Pour optimiser la recherche des voisins, il s'appuie sur l'algorithme *kd_tree*, une structure de données efficace pour réduire le temps de calcul dans des espaces à haute dimension. De plus, le paramètre *n_jobs=-1* est utilisé pour paralléliser les calculs sur tous les cœurs disponibles, accélérant ainsi à la fois l'entraînement et les prédictions.

CART

***CART* (Classification and Regression Tree)** est un modèle basé sur un arbre de décision. On utilise le critère de Gini pour mesurer l'impureté des nœuds et n'impose pas de limite à la profondeur de l'arbre, permettant ainsi d'apprendre des relations complexes.

Random Forest

Random Forest (RF) combine plusieurs arbres de décision indépendants pour améliorer la robustesse et la précision des prédictions. Ce modèle construit 100 arbres de décision sans limiter leur profondeur et utilise la méthode de *bagging* pour réduire les risques de surapprentissage. L'utilisation de tous les cœurs disponibles accélère considérablement l'entraînement.

XGBoost

***XGBoost* (Extreme Gradient Boosting)** est un modèle d'ensemble basé sur le boosting d'arbres de décision. Avec 100 arbres de décision, un taux d'apprentissage de 0,1 et une profondeur maximale de 6. L'utilisation de *logloss* comme métrique permet d'évaluer la qualité des prédictions probabilistes.

MLP

MLP* (Multilayer Perceptron)** a également été mis en place. Ce réseau de neurones artificiels comprend une couche cachée de 100 neurones, utilise la fonction d'activation ***ReLU et l'optimiseur adaptatif Adam. Il est conçu pour s'entraîner sur un maximum de 200 itérations.

Performances pour la classification binaire (`Label_n`)

Métriques

Pour des métriques de la classification binaire, tous les modèles ont obtenu une précision élevée de plus de 95 %, démontrant une bonne capacité à minimiser les faux positifs.

Cependant, le rappel est plus modéré et varie entre 56 et 59 %, ce qui reflète une performance moyenne dans la détection des attaques. L'*accuracy* des modèles est d'environ 82%, avec un *F1-score* autour de 72 %, montrant un compromis raisonnable entre précision et rappel.

Le *balanced accuracy* est d'environ 78% indique une meilleure prise en compte du déséquilibre des classes. Le *MCC*, à environ 0,64 souligne une corrélation modérée entre les prédictions et les étiquettes réelles.

Ce sont les modèles *XGBoost* et *MLP* ont montré les meilleures performances.

Les performances de nos modèles *KNN* et *Random Forest* surpassent le *KNN* et le *Random Forest* de l'article.

Temps

En termes de temps, les modèles *KNN* et *MLP* se comportent de manière totalement opposée :

- *KNN* nécessite très peu de temps pour s'entraîner, mais beaucoup de temps pour effectuer des prédictions.
- *MLP*, à l'inverse, demande beaucoup de temps pour l'entraînement, mais est très rapide lors des prédictions.

Les modèles basés sur des arbres de décision, *CART* et *XGBoost*, sont les plus rapides, que ce soit pour l'entraînement ou les prédictions.

Random Forest, de son côté, prend plus de temps car il combine les résultats de nombreux arbres souvent plus profonds, ce qui allonge les phases d'entraînement et de prédiction.

Globalement, l'entraînement du modèle *MLP* est significativement plus long et le modèle *KNN* présente des temps de prédiction plus longs.

Que ce soit pour la classification binaire ou multi-classes, les performances restent globalement similaires.

Mémoire

En termes de mémoire, c'est à nouveau *XGBoost* qui est leader, avec une consommation mémoire largement inferieure de tous les autres modèles. Et c'est encore *KNN* et *MLP* qui ont la consommation de ressources la plus élevée ainsi que Random Forest.

Conclusion

Pour la classification binaire, **XGBoost** est clairement le meilleur choix. Il combine excellentes performances, rapidité, et faible consommation de mémoire.

MLP, bien qu'il soit performant sur les métriques, est l'un des moins efficaces en termes de gestion des ressources, ce qui le rend moins adapté si la mémoire ou le temps sont des contraintes importantes.

Performances pour la classification multi-classe (`Label`)

Pour la classification multi-classe, les résultats dépendent du type d'attaque.

DoS

Lors de l'analyse exploratoire, nous avons observé que les caractéristiques de ce type d'attaque sont souvent bien distinctes des autres (par exemple, un nombre de paquets significativement plus élevé). Cela rend leur distinction relativement facile par rapport aux autres types d'attaques.

Les performances des modèles sont exceptionnelles, avec toutes les métriques dépassant 0,96. Cette fois, c'est le modèle *Random Forest* qui s'est révélé le plus performant.

Physical fault

Les attaques de type *physical fault* ne se distinguent pas des données normales en termes de caractéristiques réseau, ce qui rend leur détection quasiment impossible pour les modèles dans une classification multiclasse.

Pour *KNN*, *CART*, et *Random Forest*, la précision est d'environ 0,2, avec des *F1-scores* et des rappels proches de zéro. Les modèles *XGBoost* et *MLP* rencontrent encore plus de difficultés face à ce type d'attaque, avec la majorité des métriques réduites à 0.

MITM

Les résultats de la classification multiclasse montrent que tous les modèles testés (*KNN*, *CART*, *Random Forest*, *XGBoost*, et *MLP*) présentent des performances limitées pour détecter l'attaque *MITM*, principalement en raison de sa forte similarité avec les données normales. Bien que certains modèles, comme *XGBoost* et *MLP*, atteignent une précision globale élevée (supérieure à 0,94) et un taux de vrais négatifs (*TNR*) parfait (1,000), leurs rappels extrêmement faibles (inférieurs à 0,03) mettent en évidence leur incapacité à différencier efficacement *MITM* des données normales.

Les faibles valeurs du *F1-score* (entre 0.01 et 0.05) et de la *balanced accuracy* (~0,5) confirment la difficulté des modèles à distinguer *MITM* des autres classes.

Anomaly

Les modèles détectent parfaitement la classe *anomaly*, avec un rappel et une *balanced accuracy* de 1, malgré sa faible représentation dans le jeu de données. Cependant, une précision modérée (0.66) indique la présence de faux positifs, où des échantillons d'autres classes sont parfois prédits comme *anomaly*. C'est grâce à l'utilisation de l'*oversampling* que tous les modèles ont pu reconnaître le type d'attaque *anomaly*, malgré le fait que le jeu de données de test ne contient que deux entrées de ce type.

Conclusion

Nous avons obtenu des modèles concluants, qui surpassent même certains résultats de l'article, notamment pour certaines configurations. Les données physiques se sont avérées plus faciles à gérer, offrant de meilleurs résultats, tandis que les données réseau ont nécessité plus de nettoyage et de traitement. Les jeux de données étant très différents, cela a conduit à des résultats très différents : *XGBoost* et *MLP* se sont montrés performants pour les données réseau, mais moins efficaces pour les données physiques.

Il n'existe pas de solution "toute faite" : nous avons adopté une approche incrémentale, en testant différentes options et en ajustant plusieurs hyperparamètres pour identifier les meilleures solutions. La préparation des données a joué un rôle crucial, impactant directement les performances des modèles. Bien que nous ayons exploré un large éventail d'hyperparamètres, il est possible qu'il en existe d'autres à tester pour optimiser davantage les résultats.

Dans la plupart des cas, l'utilisation des ressources plus légères (plus rapides, moins de mémoire utilisée) ont été obtenues au détriment de la performance sur différentes métriques, et inversement. Il est donc essentiel de trouver un équilibre pour la mise en production, entre rapidité, mémoire, et précision (ou toute autre métrique qui a de l'importance dans le cas étudié).

Bien que notre analyse soit détaillée, nous recommandons de consulter notre webapp Streamlit pour une visualisation plus claire et intuitive.

Si nous avions eu plus de temps, nous aurions cherché à relier les deux jeux de données et exploré d'autres optimisations possibles. Malgré cela, nous avons appliqué des techniques de réduction de dimensions et respecté les bonnes pratiques de reproductibilité, telles que la fixation de la seed et l'oversampling/undersampling lorsque cela était pertinent. Pour les données physiques, nous avons exploré l'hypothèse de *KNN* avec *PCA*, mais nous n'avons pas testé l'application de *PCA* avec d'autres modèles.

Contributions personnelles / Partage des tâches

Zoé Marquis

Résumé de mes contributions

- J'ai pris en charge la préparation des données physiques, incluant le nettoyage des données et la réalisation d'une analyse exploratoire des séries temporelles.
- J'ai conçu et implémenté un modèle **CNN 1D** pour l'analyse des données physiques. Après avoir préparé les données pour qu'elles aient la forme correspondante à l'entrée d'un CNN 1D, j'ai ajusté les hyperparamètres du modèle, optimisé son entraînement et évalué ses performances en utilisant les différentes métriques de classification.
- J'ai mis en place la WebApp Streamlit pour offrir une exploration interactive des données et des modèles. J'ai développé l'ensemble des pages dédiées à l'analyse exploratoire des données physiques, aux matrices de confusion, aux différentes métriques de performance et leurs visualisations comparatives pour chaque modèle. J'ai également créé une page spécifique pour analyser les ressources consommées par chacun de ces modèles.
- J'ai également pris en charge l'environnement de travail, en configurant le dépôt GitHub, l'organisant de manière structurée et en configurant la base de données PickleShare pour faciliter l'intégration avec Streamlit et organiser les modèles de manière optimale. Dans le cadre de l'organisation du projet, j'ai découpé les tâches en sous-tâches afin de faciliter la prise en main du projet par l'ensemble des membres de l'équipe. Cela a permis une gestion plus fluide et une meilleure répartition des responsabilités.
- Chaque membre de l'équipe étant responsable de la rédaction de la section correspondant à son travail, j'ai ainsi rédigé les informations concernant mes tâches.

Points à retenir

- L'implémentation de Streamlit dans ce projet a été une expérience enrichissante, et cet outil pourrait certainement être réutilisé dans notre projet de Master. Il offre une valeur ajoutée en matière d'interactivité et d'exploration des données. Bien que j'aie déjà de l'expérience en visualisation de données, cette expérience m'a permis de découvrir de nouveaux outils et d'enrichir ma boîte à outils.
- Face à des ensembles de données volumineux, j'ai compris l'importance de trier et d'optimiser les données afin d'éviter de traiter des informations inutiles, tout en maximisant l'exploitation des données pertinentes. L'art de faire le tri et d'optimiser sans perdre d'informations essentielles s'avère primordial.
- Dans ce projet, j'ai également appris que face à des défis complexes, il n'existe pas de solutions universelles ou de vérités absolues. L'approche est souvent expérimentale, et il est essentiel de tenter différentes stratégies, même si elles ne semblent pas évidentes au premier abord. L'important est d'adopter une attitude proactive, de tester, d'ajuster et d'apprendre au fur et à mesure des essais, car ce processus itératif est clé pour trouver une solution optimale.

Charlotte Kruzic

Parties sur lesquelles j'ai travaillé :

- Préparation des données réseau
- Analyse exploratoire des données réseau
- Réduction de dimensions des données réseau avec l'utilisation de la PCA
- Développement de la page streamlit pour présenter les étapes de préparation et d'exploration des données réseau
- Rédaction des parties du rapport sur les tâches ci-dessus

Enseignements de ce projet :

Gestion de volumes de données importants

- Optimisation du stockage des données dans les variables
- Réflexion sur les traitements trop longs pour ce type de données
- Réduction de la taille des jeux de données tout en gardant les informations
- Utilisation de la réduction de dimension

Importance de la présentation des données

- Découverte de streamlit pour présenter les résultats
- Découverte de Plotly, également vu en TP

Importance de prendre en compte le type des données, par exemple le traitement des ports ou des enregistrements dupliqués, spécifiques au domaine réseau

Importance de la préparation des données, et de leur prétraitement pour qu'elles soient utilisables de façon optimale par les modèles

Daniil Kudriashov

Résumé de mes contributions

➢ J'ai implémenté plusieurs modèles de Machine Learning pour l'analyse des données physiques :

- KNN : Configuration optimale du nombre de voisins et de la métrique de distance
- Random Forest : Ajustement du nombre d'arbres et de leur profondeur
- XGBoost : Optimisation des hyperparamètres comme le taux d'apprentissage
- CART : Configuration de l'arbre de décision avec critère de division approprié

➢ J'ai réalisé une analyse approfondie par PCA des données physiques :

- Étude des corrélations entre variables
- Détermination du nombre optimal de composantes
- Visualisation et interprétation des résultats de la PCA
- Analyse de la contribution des variables aux composantes principales

➢ J'ai développé une page interactive Streamlit dédiée à la PCA comprenant :

- Visualisations dynamiques des résultats de la PCA
- Graphiques de variance expliquée
- Heatmaps des corrélations
- Outils d'exploration des composantes principales

➢ J'ai également contribué à l'analyse comparative des performances des différents modèles :

- Évaluation des métriques de classification
- Analyse des temps d'exécution et de la consommation mémoire

Points à retenir

➢ L'implémentation de différents modèles m'a permis de mieux comprendre leurs forces et faiblesses respectives. J'ai particulièrement apprécié pouvoir comparer empiriquement leurs performances sur un même jeu de données.

➢ L'utilisation de la PCA s'est révélée très instructive pour comprendre la structure sous-jacente des données et identifier les variables les plus importantes pour la détection d'attaques.

➢ Le développement avec Streamlit m'a fait découvrir un outil puissant pour créer rapidement des interfaces de visualisation interactives et professionnelles.

➢ Ce projet m'a permis de développer une approche méthodique de l'analyse de données, depuis le prétraitement jusqu'à l'évaluation des modèles, en passant par l'exploration et la visualisation.

Ekaterina Zaitceva

Éléments sur lesquelles j'ai travaillé :

- Préparation des données pour l'entraînement et le test des modèles, incluant la réduction de la taille du jeu de données et oversampling des classes minoritaires
- Implementation des modeles KNN, CART, Random Forest, XGBoost et MLP, avec parallélisation pour KNN et RF pour utiliser tous les coeurs disponibles d'un ordinateur
- Rédaction des parties du rapport correspondants à mes tâches

Enseignements de ce projet :

- **Importance de la gestion des classes minoritaires :** Le projet a mis en évidence les défis liés à la reconnaissance des classes minoritaires. Des techniques adaptées, telles que l'oversampling, l'undersampling ou l'ajustement des poids de classe, sont cruciales pour équilibrer les performances des modèles.
- **Complexité des classes ressemblantes :** La difficulté à distinguer certaines classes (par exemple, MITM et les données normales) souligne l'importance d'analyser les similitudes entre les classes. Cela peut nécessiter une ingénierie de caractéristiques avancée, ce qui peut être une voie d'amélioration.
- **Comprendre la nature des données pour des analyses fiables :** Avec ma collègue de partie réseaux, nous avons passé beaucoup de temps à discuter des particularités des données réseau, leur structure et leurs relations complexes. Cela a permis de déduire les meilleures approches pour préparer les données.
- **Limites des métriques globales :** Les métriques traditionnelles, telles que l'accuracy, ne reflètent pas toujours la capacité réelle des modèles à gérer les classes d'intérêt. En revanche, des métriques comme le rappel, le F1-score, et la balanced accuracy se sont avérées plus pertinentes pour évaluer les performances sur les classes minoritaires.