MACHINE LEARNING PROJECT

Paul Escalier
Victor Gelineau
Hubert Durand-Smet
Thibault Germain

Lien Colab :

https://colab.research.google.com/drive/1XRCUZB5xrvOHg1Cp0BKpfe_W3PeADJoJ?usp=sharing

Rappel du contexte- :

Lors des premières étapes, nous avions sélectionné des datasets intéressants, que nous avions ensuite analysés, décrits, et nettoyés en identifiant les éléments pertinents. Ce travail préliminaire a été réalisé sur un premier dataset. Nous avons ensuite essayé à plusieurs reprises d'utiliser diverses solutions d'intelligence artificielle pour prédire certaines colonnes en se basant sur des variables sélectionnées en entrée. Malgré ces tentatives, comprenant l'utilisation de multiples algorithmes et combinaisons de variables, aucune méthode n'a permis d'obtenir des prédictions satisfaisantes en raison de l'absence de corrélations significatives.

Face à ce constat, nous avons opté pour un second dataset. Sur ce dernier, nous avons réalisé des visualisations et des analyses de corrélations qui ont abouti à la conception d'un algorithme, issu des enseignements du cours, permettant de prédire efficacement avec un taux de réussite satisfaisant. Nous avons donc pu identifier les différents problèmes liés à notre ancien dataset. Tel que l'absence de liens entre certaines de nos variables, ce qui nous a permis de consacrer davantage de temps à l'analyse de ces données sur ce nouveau dataset.

I- Analyse et explication des données du dataset :

Description globale du dataset NSL-KDD et de ses variables

a) Description Globale

Le dataset NSL-KDD est un benchmark utilisé pour l'évaluation des systèmes de détection d'intrusions (IDS). Il est dérivé du dataset KDD Cup 1999, mais a été amélioré pour corriger des biais tels que la redondance excessive et l'importance disproportionnée de certaines instances. Ce dataset est organisé en plusieurs variables qui capturent des caractéristiques

essentielles du trafic réseau, incluant des informations contextuelles, des métriques de performance réseau, et des indicateurs d'activité potentiellement malveillante.

Chaque observation représente une connexion réseau caractérisée par plusieurs attributs mesurant des aspects variés comme la durée de la connexion, le type de protocole, le nombre d'erreurs, ou encore les volumes de données échangées. Ces variables sont associées à une étiquette binaire (`attack`) indiquant si une intrusion est détectée.

b) description des variables

Le dataset NSL-KDD comprend 43 variables qui décrivent des connexions réseau et permettent de détecter des activités malveillantes. Ces variables se regroupent en plusieurs catégories clés :

1. Caractéristiques générales des variables liés à la connexion

Les variables comme duration (durée de la connexion), protocol_type (protocole utilisé, ex. TCP, UDP) et service (type de service réseau, ex. HTTP, FTP) décrivent le contexte technique des connexions. La variable flag indique l'état des drapeaux TCP après une connexion, fournissant des informations sur le statut des sessions établies. Ces caractéristiques sont essentielles pour identifier des comportements réseau normaux ou inhabituels.

2. Statistiques de trafic

Des mesures comme src_bytes (données envoyées par la source) et dst_bytes (données reçues par la destination) fournissent une vue sur les volumes de trafic. Les variables count et srv_count comptent respectivement le nombre de connexions observées pour un hôte source ou un service spécifique dans une fenêtre temporelle. Ces statistiques aident à détecter des schémas tels que des attaques par déni de service ou des balayages de ports.

3. Indicateurs d'état et d'erreurs

Les variables telles que serror_rate (taux d'erreurs SYN), rerror_rate (taux de connexions rejetées) et wrong_fragment (nombre de fragments malformés) renseignent sur l'état des connexions et les anomalies réseau. Ces indicateurs permettent de détecter des activités suspectes comme des attaques SYN flood ou des communications malveillantes basées sur des paquets anormaux.

4. Variables concernant les activités spécifiques liées à la sécurité

Plusieurs variables surveillent des actions liées à la sécurité, telles que num_failed_logins (tentatives de connexion échouées), root_shell (obtention d'un accès root), num_file_creations (création de fichiers) et num_shells (ouverture de shells). Ces indicateurs signalent des intrusions potentielles, des escalades de privilèges ou des activités inhabituelles ciblant des ressources critiques.

5. Variables d’hôte distant

Des variables comme dst_host_count (nombre de connexions vers un hôte cible) et dst_host_srv_count (connexions vers un même service sur un hôte cible) permettent d’identifier des comportements anormaux au niveau des cibles réseau.

Les taux associés, comme dst_host_same_srv_rate ou dst_host_srv_rerror_rate, mesurent les proportions d’activités sur ces cibles et aident à détecter des schémas malveillants comme des attaques ciblées ou distribuées.

6. Variables cibles et contextuelles

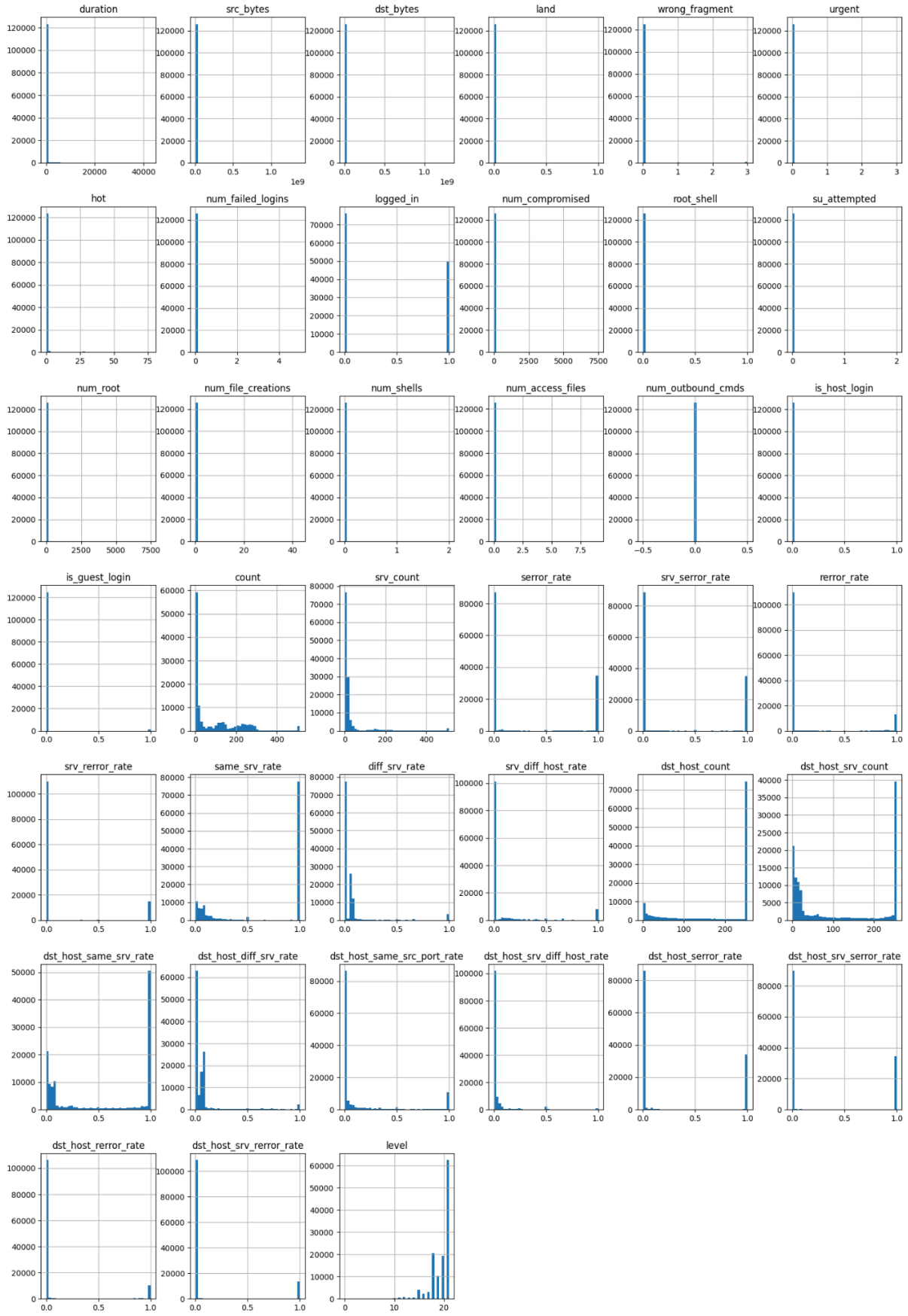
La variable attack sert de cible principale, indiquant si une connexion est légitime (0) ou malveillante (1). La variable level quantifie la gravité des attaques, avec des valeurs plus élevées signalant des intrusions sophistiquées ou dangereuses.

Synthèse

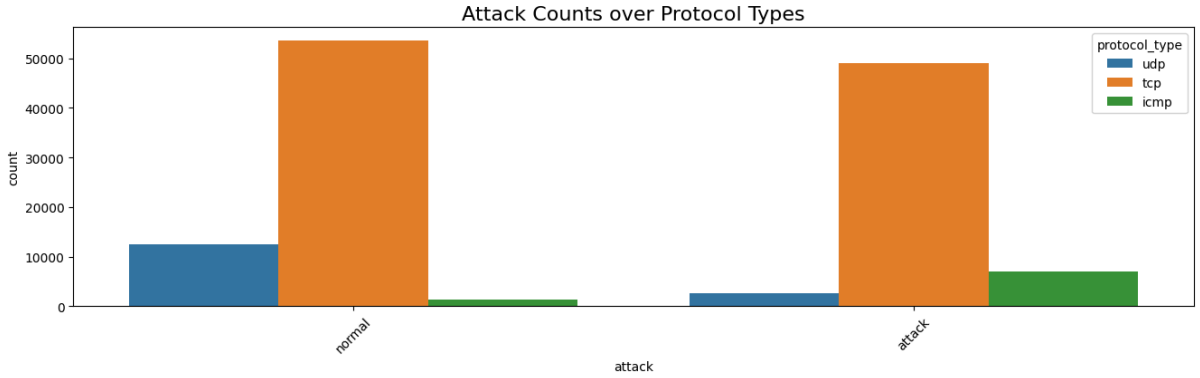
Ces catégories regroupent des variables descriptives, comportementales et sécuritaires qui offrent une vue multidimensionnelle des connexions réseau. Grâce à cette richesse, le dataset NSL-KDD est un outil puissant pour entraîner et évaluer des modèles capables de détecter des menaces variées avec précision.

Le dataset regroupe 43 variables, réparties en catégories ce qui offre un large choix concernant les variables à prendre pour les modèles.

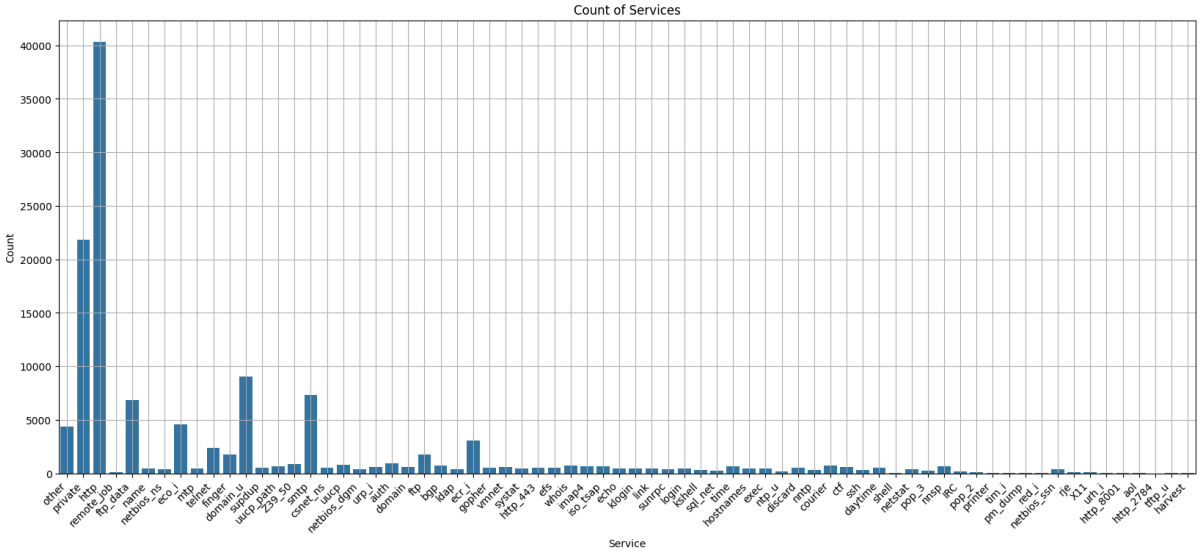
c) Visualisation des caractèristiques de ces variables :
   Nous avons ainsi fait différentes visualisations pour observer ces variables :



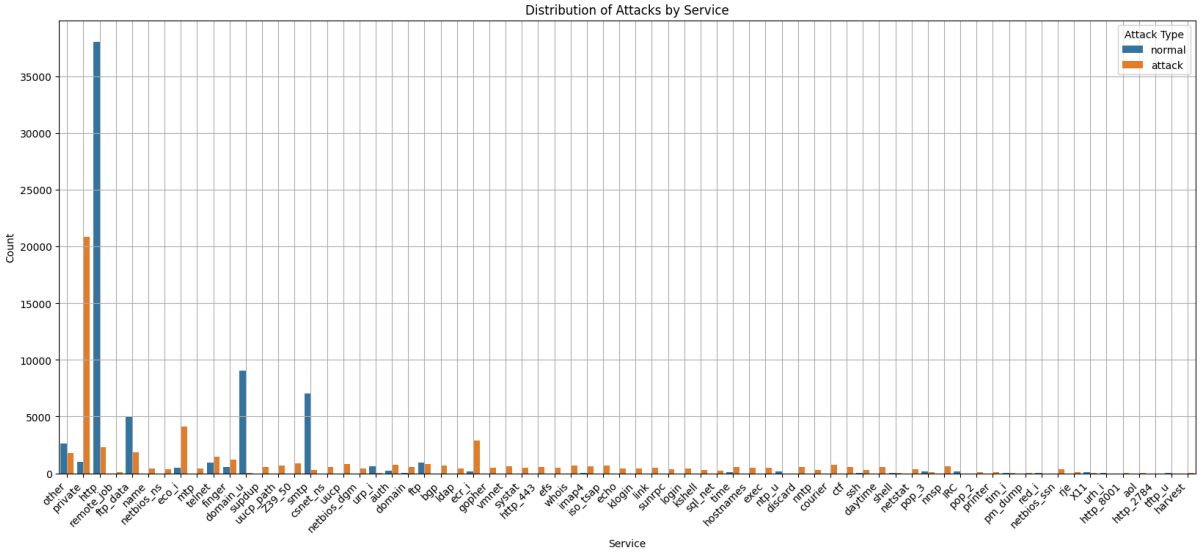
Visualisation générale pour obtenir la répartition des variables sur chaque valeurs



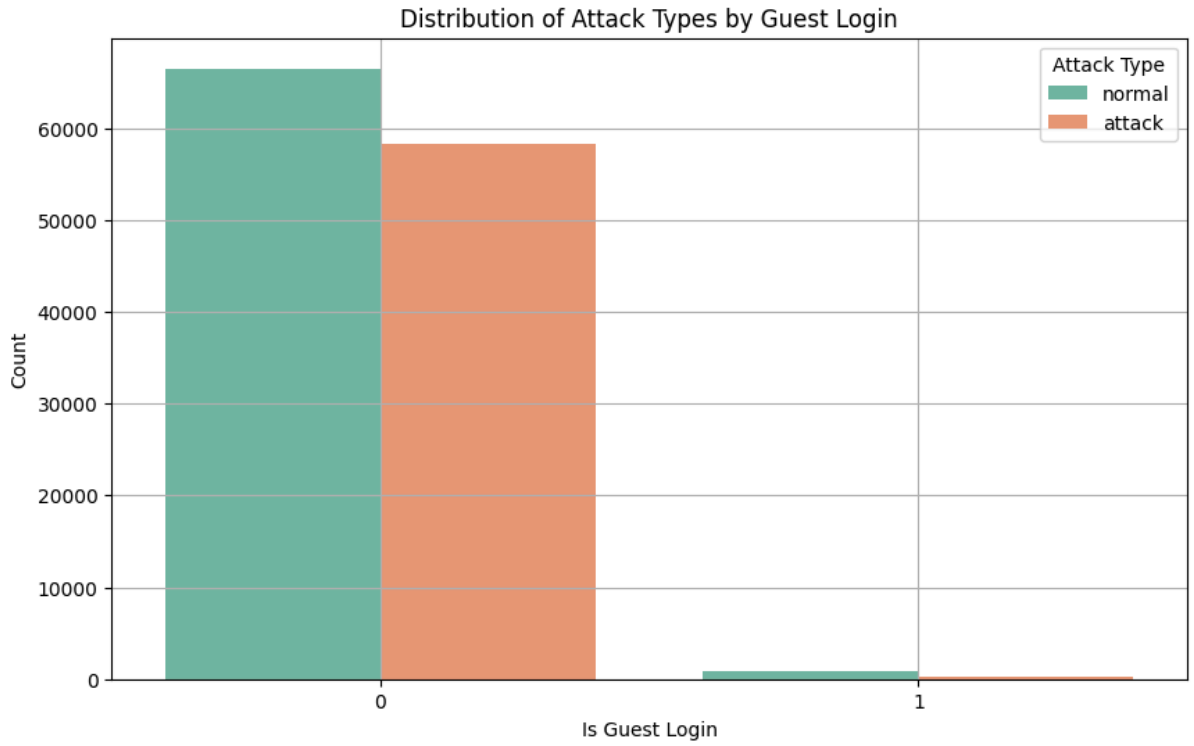
Nous pouvons donc voir que la plupart des attaques proviennent de TCP, puis d'UDP, et que moins d'attaques proviennent de ICMP.



Les services les plus utilisés en général sont les suivants : http, privé, domaine_u, smtp, ftp, autre



nous pouvons voir que les attaques privées sont le service le plus courant



nous pouvons clairement dire que les attaques surviennent lorsque l'invité n'est pas connecté

La quantité de variables fut, dans un premier temps, assez compliquée à comprendre. Nous avons donc effectué des analyses et surtout posé pour comprendre le dataset et ce qu'il impliquait.

II - Explication des IA

Après avoir analysé les données et trouvé notre jeu de données pertinent et surtout exploitable, nous avons décidé de procéder à des tests de prédiction.

Dans un premier temps, nous avons dû transformer certaines variables, jugées potentiellement utiles, mais qui n'étaient pas au format numérique. Ces variables sont :
`clm = ['protocol_type', 'service', 'flag', 'attack']`.
Nous les avons donc converties en variables numériques, facilitant ainsi leur interprétation ultérieure.

Nous avons ensuite réfléchi à la variable la plus pertinente à prédire. Nous avons choisi la variable `attack` pour une première analyse, car sa nature binaire est intéressante.

Dans un second temps, nous avons entraîné notre modèle d'IA sur deux axes : l'axe X comprenant toutes les colonnes sauf celle de `attack`, et l'axe Y étant la colonne `attack`. Nous avons ainsi pu analyser la corrélation entre `attack` et chacune des autres variables, ce qui a donné les résultats suivants :

```
from sklearn.feature_selection import mutual_info_classif
mutual_info = mutual_info_classif(X_train, y_train)
mutual_info = pd.Series(mutual_info)
mutual_info.index = train_index
mutual_info.sort_values(ascending=False)
```

	0
src_bytes	0.533482
dst_bytes	0.482106
service	0.332252
dst_host_srv_count	0.240379
flag	0.237106
dst_host_same_srv_rate	0.234493
dst_host_diff_srv_rate	0.230167
dst_host_rerror_rate	0.211444
count	0.184161
diff_srv_rate	0.180825
dst_host_srv_rerror_rate	0.180751
same_srv_rate	0.171821
level	0.169707
rerror_rate	0.160458
logged_in	0.156476
srv_rerror_rate	0.152106
dst_host_srv_diff_host_rate	0.134869
dst_host_same_src_port_rate	0.118576
srv_diff_host_rate	0.101086
dst_host_count	0.088404

Donc ici nous avons sur cette image juste une partie de la corrélation de chaque variable, Ici on voit des variables corrélés à df attack, chacun à son échelle. Ainsi ici on observe que attack n'est pas corrélés à une variable.

Ensuite nous utilisons la bibliothèque **Scikit-learn** pour sélectionner les variables les plus pertinentes parmi celles disponibles dans un jeu de données, en fonction de leur relation

avec la variable cible. Nous utilisons cette technique pour réduire la dimensionnalité et améliorer les performances d'un modèle de prédiction.

```
[ ] from sklearn.feature_selection import SelectKBest
    Select_features = SelectKBest(mutual_info_classif, k=30)
    Select_features.fit(X_train, y_train)
    train_index[Select_features.get_support()]
```

```
Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'wrong_fragment', 'hot', 'logged_in', 'is_guest_login',
       'count', 'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate',
       'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
       'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count',
       'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
       'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
       'dst_host_serror_rate', 'dst_host_srv_serror_rate',
       'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'level'],
      dtype='object')
```

```
columns=['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'wrong_fragment', 'hot', 'logged_in', 'num_compromised',
       'count', 'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate']

#We will continue our model with top 15 features, because dataset is big enough
```

Puis nous avons normalisé nos données avec fit
Ainsi en résumé nous avons sélectionné nos 15 colonnes les plus corrélées avec la variable attack (normal si pas d'attaque, autre s'il y a une attaque). Nous avons par la suite vérifié manuellement la logique derrière l'utilisation des variables

.
Nous n'avions plus qu'à essayer différents types d'algorithmes:

Logistic Regression

Pourquoi l'avoir utilisé ?

Avantage :

Simplicité :

- Facile à implémenter.
- Convient bien pour des problèmes où les relations entre les variables sont linéaire (ce qui semble être le cas ici)

Rapidité et efficacité :

- Fonctionne rapidement sur des jeux de données de taille moyenne.

- Consomme moins de ressources en termes de calcul par rapport à des modèles plus complexes comme les arbres de décision ou les réseaux neuronaux.

Résilience au sur-apprentissage :

- Avec une régularisation appropriée (par exemple, L1 ou L2), la régression logistique peut limiter les risques de sur-apprentissage (overfitting).

Inconvénients :

- Si les relations sont non linéaires, les performances peuvent être médiocres Ici nos relations sont linéaires et donc non dérangeant

Sensible aux outliers :

- Les valeurs aberrantes peuvent affecter significativement les coefficients. Ici nous avons l'avantage que notre dataset était déjà en partie nettoyé de ces valeurs aberrantes. Donc moins touché par ce fait.

Performances sur des jeux de données déséquilibrés :

- Peut avoir des performances médiocres si les classes sont déséquilibrées, bien que cela puisse être corrigé avec des pondérations ou un échantillonnage. Voici une des raisons de pourquoi nous avons normalisé !

Résultat : Environ 89%

Ainsi, il s'agit d'un résultat qui nous indique, ainsi une plutôt bonne probabilité de résultat. C'est assez intéressant, ici cet algorithme semble convenir assez clairement à notre dataset ce qui explique son bon résultat. Ce qui explique, les légères erreurs peuvent être potentiellement le surapprentissage, que nous avons tenté d'éliminer, en sélectionnant seulement les valeurs les plus corrélées. Ou bien par le fait que certaines variables ne soient corrélés directement à cette variable non pas par le fait d'eux même mais directement d'une dépendance à une autre variable.

DecisionTreeClassifier

Avantages:

1. **Facile à interpréter :**
    - Les arbres de décision sont simples à visualiser et à comprendre. Chaque nœud correspond à une règle décisionnelle qui peut être expliquée en termes clairs.
    - Pour des datasets complexes, il est facile d'expliquer pourquoi une prédiction a été faite.
2. **Capable de gérer des données mixtes :**

    - Les arbres de décision peuvent traiter à la fois des caractéristiques numériques (comme duration) et catégoriques (comme protocol_type), bien que dans notre cas les catégories aient été encodées.
3. **Pas besoin de normaliser ou standardiser les données :**
    - Contrairement à certains modèles comme la régression logistique ou SVM, les arbres de décision n'exigent pas de mise à l'échelle des données (comme avec StandardScaler).
4. **Flexible aux interactions non linéaires :**
    - Les arbres capturent facilement les interactions complexes entre les caractéristiques sans nécessiter de transformations manuelles.
5. **Gestion robuste des valeurs manquantes :**
    - Les arbres de décision peuvent gérer des données incomplètes sans nécessiter un prétraitement intensif.
6. **Faible coût de prédiction :**
    - Une fois construit, l'arbre est rapide à utiliser pour faire des prédictions.

Inconvénients

Overfitting (sur-apprentissage) :

- Les arbres de décision simples ont tendance à s'ajuster excessivement aux données d'entraînement, surtout si l'arbre est profond. Cela peut entraîner une faible généralisation sur les données de test.

Sensible aux variations des données :

- Les arbres de décision sont instables. Une petite modification dans les données d'entraînement peut entraîner une structure d'arbre complètement différente.

Complexité dans les datasets larges :

- Si notre dataset contient un grand nombre de caractéristiques et/ou de classes, l'arbre peut devenir extrêmement complexe et difficile à interpréter.

Pas le meilleur choix pour les datasets déséquilibrés :

- Les arbres de décision simples peuvent être biaisés envers les classes majoritaires, à moins d'appliquer des techniques d'équilibrage (comme un rééchantillonnage ou un ajustement des poids).

Moins performant que les modèles ensemblistes :

- Un simple arbre de décision est souvent surpassé par des modèles ensemblistes comme Random Forest, AdaBoost ou XGBoost.

Ne capture pas bien les relations linéaires :

- Les arbres de décision sont moins efficaces pour des relations strictement linéaires entre les caractéristiques et la cible.

Résultat : Environ 0.97 ce qui est très bon, mais peut être inquiétant. Il y a différentes raisons à cela. Dans un premier temps, c'est un dataset qui semble très adapté, mais peut être que cette précision a tourné en notre faveur ! Potentiellement, une erreur qu'en dépit de nos vérifications nous n'avons pas su trouver (Overfitting, Dataset déséquilibré, Dataset trop simple ou trop petit, Mauvaise métrique utilisée)

modèle SVM avec un noyau linéaire

Avantages

Efficace pour les données linéairement séparables :

- Si notre dataset est linéairement séparable dans son espace actuel (ou après transformation), un SVM avec un noyau linéaire peut fournir de très bonnes performances.

Robuste au surapprentissage :

- SVM utilise un **marge maximale** pour séparer les classes, ce qui le rend moins sujet à l'overfitting, surtout lorsque le nombre de caractéristiques dépasse le nombre d'exemples.

Convient pour les grandes dimensions :

- Le SVM fonctionne bien avec des jeux de données comportant de nombreuses caractéristiques, même si celles-ci sont plus nombreuses que les exemples (par exemple, les données textuelles).

Flexibilité grâce à la régularisation :

- L'hyperparamètre **C** permet de contrôler la complexité du modèle en équilibrant la maximisation de la marge et la minimisation des erreurs d'entraînement.

Utilise peu de mémoire lors des prédictions :

- Une fois le modèle ajusté, seuls les **vecteurs supports** sont nécessaires pour prédire, ce qui rend SVM efficace pour les prédictions.

Inconvénients

Nécessite des données bien prétraitées :

- Le SVM est sensible à l'échelle des données. Vous avez déjà utilisé **StandardScaler**, ce qui est nécessaire pour éviter que des caractéristiques ayant des échelles différentes ne dominent les autres.

Moins performant sur des datasets déséquilibrés :

- Si notre dataset est déséquilibré (une classe ayant beaucoup plus d'échantillons que l'autre), le SVM peut être biaisé. Cela peut être corrigé en ajustant les poids des classes avec l'argument class_weight='balanced'.

Pas efficace pour les grands datasets :

- Le SVM avec noyau linéaire peut être lent sur des datasets très volumineux (en termes de nombre d'exemples). Cela est dû au coût quadratique ou cubique de son optimisation.

Sensibilité au bruit :

- Les SVM sont sensibles aux données bruitées et aux points aberrants, car ces derniers peuvent influencer les vecteurs supports et donc la frontière de décision.

Pas directement interprétable :

- Contrairement à un modèle comme les arbres de décision, les SVM ne sont pas directement interprétables. Il peut être difficile de comprendre pourquoi une certaine prédiction a été faite.

Résultat: 92%

L'algorithme **Naive Bayes**

Pourquoi utiliser Naive Bayes ?

Pourquoi choisir l'algorithme Naive Bayes ?

Naive Bayes est particulièrement adapté à des cas spécifiques tels que la classification textuelle, où il excelle dans des tâches comme le filtrage d'e-mails (spam ou non-spam), l'analyse de sentiments (avis positifs ou négatifs) ou la catégorisation de documents. Il est également utilisé dans des domaines médicaux et biologiques, par exemple pour diagnostiquer certaines maladies ou classer des séquences ADN. De plus, cet algorithme est souvent privilégié lorsqu'il s'agit de résoudre des problèmes avec peu de données ou un grand nombre de variables indépendantes.

En termes de performance, Naive Bayes affiche une précision d'environ 79 %. Cette précision, bien que plus faible que celle de modèles plus complexes, s'explique par les hypothèses simplificatrices sur lesquelles repose l'algorithme, comme l'indépendance des

variables. Dans notre cas, ces hypothèses ne conviennent pas totalement à notre dataset, notamment en raison du faible volume de données disponibles.

Par rapport à d'autres modèles comme les machines à vecteurs de support (SVM) ou les réseaux neuronaux, Naive Bayes se distingue par sa rapidité et ses faibles besoins en ressources computationnelles. Cependant, après avoir analysé différents algorithmes en fonction des spécificités de notre dataset, nous avons finalement opté pour les SVM, qui offrent des performances plus adaptées à notre problématique.

Les Machines à Vecteurs de Support (SVM) sont souvent considérées comme l'un des meilleurs algorithmes pour les tâches de détection d'intrusion sur des jeux de données comme le NSL-KDD en raison de plusieurs avantages :

1. **Haute Précision** : Les SVM sont connus pour leur capacité à traiter efficacement les problèmes de classification linéaire et non linéaire. Des études montrent que les SVM offrent de solides performances dans la détection des intrusions réseau, surpassant souvent d'autres classificateurs traditionnels lorsqu'ils sont associés à des méthodes appropriées de sélection de caractéristiques.
2. **Efficacité avec des Données à Haute Dimension** : Les SVM fonctionnent bien dans des scénarios avec un grand nombre de caractéristiques, comme dans les jeux de données de détection d'intrusion. En utilisant des astuces de noyau, ils peuvent trouver efficacement les frontières de décision, même dans des espaces de caractéristiques complexes.
3. **Généralisation aux Données Inconnues** : Les SVM se concentrent sur la maximisation de la marge entre les classes, ce qui améliore leur capacité à se généraliser sur de nouvelles données non vues. Cette propriété est particulièrement cruciale dans la détection d'intrusions, où de nouveaux types d'attaques peuvent ne pas avoir fait partie des données d'entraînement.
4. **Résilience au Bruit** : La robustesse des SVM face aux valeurs aberrantes et au bruit dans les données, comme les caractéristiques redondantes ou non pertinentes, en fait un choix adapté pour des jeux de données comme le NSL-KDD, où le bruit est un problème fréquent.
5. **Succès Démontré dans les Études** : De nombreux articles de recherche ont mis en évidence les performances des SVM sur le jeu de données NSL-KDD. Par exemple, des études comparant plusieurs algorithmes ont trouvé que les SVM se classaient systématiquement parmi les meilleurs en termes de précision et de taux de détection pour les tâches de détection d'intrusions réseau.

Ainsi, pour les raisons évoquées précédemment, nous avons choisi cet algorithme malgré son taux de précision légèrement inférieur à d'autres options. Selon notre analyse, il s'agit de celui qui s'adapte le mieux à notre dataset, offrant une base stable et durable, même en cas d'augmentation de la taille du dataset.

De plus, nous avons mené des tests avec différentes proportions entre les données de test et d'entraînement. La configuration qui s'est révélée la plus précise est celle utilisant 40 % des données pour les tests et 60 % pour l'entraînement.

Conclusion

Après une analyse approfondie sur les données de NSL-KDD, nous avons identifié les variables clés et mis en œuvre plusieurs algorithmes pour évaluer leurs performances. Parmi les modèles testés, le DecisionTreeClassifier a affiché les meilleurs résultats avec une précision de 97 %, bien que ce score puisse indiquer un risque de surapprentissage. Le SVM, avec une précision de 92 %, a montré une meilleure généralisation et une grande efficacité dans la classification, ce qui en fait notre choix final.

Ce projet souligne l'importance de la qualité du dataset et des prétraitements dans la réussite des modèles prédictifs. Malgré les limites liées au bruit et aux déséquilibres des données, nous avons démontré la pertinence d'une approche itérative combinant sélection de caractéristiques, normalisation, et évaluation rigoureuse des algorithmes.

Enfin, cette expérience a permis de mieux comprendre les défis liés à l'application des techniques de machine learning dans des contextes réels et a renforcé notre capacité à sélectionner les outils les plus adaptés à des problématiques spécifiques.

Bibliographie :

https://dataanalyticspost.com/Lexique/svm/
https://larevueia.fr/support-vector-machines-svm/
https://link.springer.com/article/10.1007/s41870-017-0080-1

https://ieeexplore.ieee.org/abstract/document/4721435
https://www.sciencedirect.com/science/article/pii/S0952197624011308

https://www.sciencedirect.com/science/article/abs/pii/S0181551213002490
https://learn.microsoft.com/fr-fr/archive/msdn-magazine/2015/april/test-run-multi-class-logistic-regression-classification
https://www.inderscienceonline.com/doi/abs/10.1504/IJDATS.2011.041335

https://www.inderscienceonline.com/doi/abs/10.1504/IJIDS.2020.108141
https://scikit-learn.org/1.5/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://medium.com/swlh/decision-trees-classifier-aba3c53e14b9#:~:text=What%20is%20the%20Decision%20Trees,both%20continuous%20and%20categorical%20problems.