

Royaume du Maroc
UNIVERSITÉ MOHAMED V - RABAT

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE
ET D'ANALISE DES SYSTÈMES



Rapport de projet Machine Learning et Deep learning

Detecting Malicious URLs

Filière : Sécurité des systèmes d'information (SSI)

Réalisé par :

AGOULZI Imane
AMELLALN Ayoube
EL BERDAI Adam
JOUIJATE Rim

Professeur :
BENBRAHIM Houda

Année universitaire : 2022 - 2023

Table des matières

Introduction générale	1
1 Présentation du projet	2
1.1 Introduction	2
1.2 Collecte de données	2
1.3 Exploration de données	3
1.4 Traitement de données	3
2 Techniques de classification	4
2.1 Algorithmes d'apprentissage supervisé :	4
2.1.1 K-nearest neighbor (Knn) :	4
2.1.2 Naive Bayes :	4
2.1.3 Arbre de décision :	5
2.1.4 Support Vector Machine (SVM) :	5
2.2 Algorithmes d'apprentissage non supervisé :	5
2.2.1 Clustering hiérarchique :	5
2.2.2 K-means :	5
2.2.3 DBSCAN :	6
2.3 Perceptron multi-couches (MLP) :	6
3 Résultats expérimentaux	7
4 Discussion	11
4.1 Apprentissage Supervisé	11
4.2 Apprentissage Non-Supervisé	11
Conclusion	12
Annexes	14
Code Complet avec Commentaires	14

Introduction générale

Le domaine de la sécurité informatique est de plus en plus important à mesure que les attaques en ligne deviennent plus fréquentes et sophistiquées. La détection des URL malveillantes en est un aspect crucial, car elle permet de protéger les utilisateurs contre les attaques de phishing et les intrusions informatiques. Dans ce projet, nous nous concentrerons sur la détection des URL malveillantes en utilisant des techniques d'apprentissage automatique.

Notre problématique est de "Comment utiliser les caractéristiques de structure de l'URL, telles que la longueur de l'URL, la longueur du hostname, la longueur du path, le nombre de '-' et le nombre de chiffres, combinées aux informations de contexte comme l'utilisation d'une IP ou d'une URL raccourcie, pour détecter les URL malveillantes à l'aide de techniques d'apprentissage supervisé, non supervisé et de réseau de neurones multi-couches ?".

Notre objectif est d'évaluer les performances des différentes techniques d'apprentissage automatique pour détecter les URL malveillantes, en utilisant un ensemble de données d'URL étiquetées. Nous testons des différents algorithmes et modèles pour identifier les caractéristiques les plus pertinentes pour la détection des URL malveillantes, et pour comparer les performances des différentes techniques.

Les résultats de nos expériences sont importants car ils montrent comment les caractéristiques de structure de l'URL et les informations de contexte peuvent être utilisées pour détecter les URL malveillantes de manière efficace. Ils peuvent également aider les chercheurs et les développeurs à améliorer les systèmes de sécurité informatique pour protéger les utilisateurs contre les attaques en ligne.

Dans la littérature existante, les chercheurs ont utilisé des techniques telles que les filtres de contenu, les signatures de malware, les analyse de réseau, l'analyse de contenu et l'analyse de comportement pour détecter les URL malveillantes. Notre approche se différencie en utilisant des techniques d'apprentissage automatique pour identifier les caractéristiques les plus pertinentes et les plus informatives à partir des données d'URL étiquetées. Nous comparons également les performances de différentes techniques d'apprentissage automatique pour détecter les URL malveillantes.

Chapitre 1

Présentation du projet

1.1 Introduction

Nous avons suivi une démarche standard en ce qui concerne la phase de pré-traitement des données. Nous commençons par le bon choix de dataset depuis celles disponibles sur Internet. Ensuite, nous allons nettoyer les données (exploration des données) pour les traiter (traitement des données) afin d'avoir un dataset près à être utilisé dans les techniques de classification.

1.2 Collecte de données

Le commencement de cette phase est marqué par notre recherche sur l'Internet de dataset convenable pour notre problématique. Nos principaux critères sont donc :

- Dataset assez grand
- Données Etiquetées
- Données bien diversifiées

Nos recherches nous ont emmenés à se basés principalement sur Kaggle [?] et etc. Nous avons éliminer les datasets dont la taille est trop grande et ceux dont les données n'étaient pas étiquetées ou étaient fortement associés à un certain attribut (contenu). Finalement, la dataset qu'on a choisi est une dont on a jugé la taille et les données convenables pour l'étude que nous allons conduire.

Notre dataset contient 450176 données étiquetées (à peu près 35Mb), dont les classes sont deux : Malveillant (23,20%) et Non-Malveillant(76,80%). Les attributs existants sont :

1. Indexe.
2. URL : contient tout l'URL.
3. Label : Spécifie la classe du URL.
4. Result : Encodage en binaire des classes.

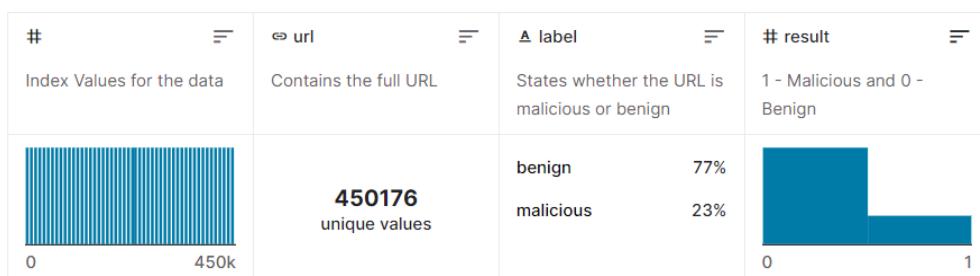


FIGURE 1.1 – Informations initiales sur le dataset utilisé

1.3 Exploration de données

Dans cette étape, nous allons adapter la dataset qu'on a trouvé à notre contexte. Nous allons d'abords éliminer les attributs qui nous ne donne pas d'information utile concernant les algorithmes de Machine Learning et Deep Learning, ainsi que les données auxquelles sont attribuées des valeurs nulles ou inutiles.

Après réalisation de ces tâches, nous nous assurons que les données restantes sont pertinentes pour répondre à notre problématique.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 450176 entries, 0 to 450175
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  -- 
 0   url      450176 non-null   object 
 1   label     450176 non-null   object 
 2   result    450176 non-null   int64  
dtypes: int64(1), object(2)
memory usage: 10.3+ MB
```

url	0
label	0
result	0
dtype:	int64

FIGURE 1.2 – Informations sur les données après exploration + l'inexistence des valeurs nulles

1.4 Traitement de données

Dans cette étape, nous allons extraire les attributs que nous jugeons pertinents pour la bonne performance des techniques de classification, ainsi que faire quelques visualisations, et davantage éliminer les attributs corrélés pour faciliter la vitesse des algorithmes, et minimiser l'usage des ressources.

Extraction des Attributs On veut extraire trois catégories d'attributs :

1. Structure de l'URL
 - la longueur d'URL
 - la longueur du Hostname
 - la longueur du path
 - la longueur du First Directory
2. Contenu de l'URL
 - nombre de '-'
 - nombre de '@'
 - nombre de '?"
 - nombre de '.'
 - nombre de '
 - nombre de '='
 - nombre de 'http'
 - nombre de chiffres
 - nombre de Lettres
 - nombre de Directories
3. Attributs binaires
 - l'utilisation de IP
 - l'utilisation de Shortened URL

Chapitre 2

Techniques de classification

La classification est une technique d'apprentissage automatique utilisée pour attribuer une étiquette de classe à un échantillon d'entrée. Elle est utilisée dans une grande variété d'applications telles que la reconnaissance d'images, le traitement du langage naturel et la détection de spam. Le but de la classification est de prédire avec précision la classe de données non vues en fonction des modèles appris à partir des données d'entraînement. Les algorithmes couramment utilisés dans la classification incluent la régression logistique, les arbres de décision et les machines à vecteurs de support.

2.1 Algorithmes d'apprentissage supervisé :

Les techniques d'apprentissage supervisé permettent, à l'aide d'un ensemble d'entraînement dont les données sont déjà étiquetées, de classifier de nouvelles données jamais vues parmi les classes qu'on a introduit dans notre ensemble d'entraînement. Une vérification est faite suivant un ensemble de test.

2.1.1 K-nearest neighbor (Knn) :

KNN signifie "k-voisins les plus proches". C'est un type d'algorithme d'apprentissage supervisé utilisé pour les problèmes de classification et de régression. L'objectif de KNN est de trouver le nombre des points les plus proches dans l'espace de présentation pour prédire la classe en fonction de la moyenne ou de la majorité de leurs classes. Pour l'application de cette algorithme sur notre dataset, on est besoin de chercher le bon k, c'est le nombre de voisins à prendre en considération. Dans notre modèle, on a pris le k par défaut de l'algorithme knn qui est la valeur 5.

2.1.2 Naive Bayes :

L'algorithme Naive Bayes est un algorithme probabiliste qui utilise la théorie de Bayes pour prédire l'étiquette de classe d'une instance en fonction de ses valeurs de caractéristiques. L'algorithme suppose que les caractéristiques sont indépendantes, donnée l'étiquette de classe, et calcule la probabilité de chaque caractéristique donnée l'étiquette de classe indépendamment. Il existe plusieurs variations de l'algorithme Naive Bayes, telles que Naive Bayes gaussien, Naive Bayes multinomial et Naive Bayes de Bernoulli, qui sont utilisées en fonction du type de données. Une fois les probabilités calculées, l'algorithme utilise la théorie de Bayes pour calculer la probabilité a posteriori de chaque étiquette de classe donnée les caractéristiques et prédit ensuite l'étiquette de classe avec la plus grande probabilité.

2.1.3 Arbre de décision :

Un arbre de décision est une structure de graphe arborescente, où chaque nœud représente un test sur un attribut. Chaque branche représente le résultat du test et les nœuds feuilles représentent l'étiquette de classe obtenue après toutes les décisions prises via cette branche. Les chemins de la racine à la feuille représentent des règles de classification. Lors de l'implémentation du modèle, de la bibliothèque scikit-learn, il y a trois paramètres qu'on peut modifier : on trouve la profondeur maximale de l'arbre, c'est par défaut non définie c'est-à-dire on s'arrête lorsqu'on est avec des feuilles seulement. Puis le nombre minimum d'échantillons requis pour se retrouver à un nœud de feuille, par défaut c'est 1. Et finalement, la fonction de mesure de la qualité d'une scission, il peut prendre soit le critère gini qui correspond à l'impureté de Gini soit entropy qui correspond au gain d'informations, et elle est par défaut gini.

2.1.4 Support Vector Machine (SVM) :

SVM est un type d'algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Il fonctionne en trouvant un séparateur optimal entre les différentes classes en utilisant des vecteurs de support. Les vecteurs de support sont les points les plus proches de la frontière de décision. L'algorithme maximise la marge, c'est-à-dire la distance entre la frontière de décision et les vecteurs de support les plus proches des classes, pour maximiser la robustesse de la classification. Dans notre algorithme, on va s'intéresser seulement aux 3 paramètres et les autres auront des valeurs par défaut, ces paramètres sont : gamma, coefficient de vitesse et kernel. Il faut bien tester des différentes valeurs pour avoir la bonne combinaison des paramètres qui donne un bon modèle.

2.2 Algorithmes d'apprentissage non supervisé :

Les algorithmes d'apprentissage non-supervisé, au contraire des algorithmes ci-dessus, ne s'intéresse pas aux étiquettage ou non des données. Plutôt, ils visent principalement à créer les étiquettes pour les données, selon un certain nombre d'étiquettes qui est généralement spécifié par l'utilisateur.

Ainsi, leur objectif est de séparer les données en des classes bien définies, de façon à ce qu'on puisse après définir les classes trouvées. De la même façon qu'avant,

2.2.1 Clustering hiérarchique :

Le clustering hiérarchique essaie de créer les clusters finals pas à pas, en introduisant à chaque fois un nouveau enregistrement dans un cluster (on dit qu'il est agglomérative, c'est la variante qu'on va utiliser). La condition d'arrêt peut être un nombre de clusters atteint, où le nombre d'itérations maximal atteint.

2.2.2 K-means :

K-means est un algorithme d'apprentissage automatique non supervisé qui est utilisé pour grouper les points de données en k cluster. L'algorithme fonctionne en initialisant d'abord aléatoirement les centroïdes du cluster k, puis en assignant de façon itérative chaque point de données au cluster avec le centroïde le plus proche et en mettant à jour le centroïde à la moyenne des points de données dans ce cluster. Ce processus est répété jusqu'à ce que les affectations du groupe ne changent plus. Pour une bonne application de cette algorithme, il faut bien choisir

le valeur du paramètre K, pour cela, on utilisera la méthode de coude (La méthode Elbow) qui va nous aider à touver le k optimale pour nos clusters.

2.2.3 DBSCAN :

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) est un algorithme de clustering basé sur la densité utilisé pour identifier des groupes de points dans un espace multidimensionnel. Il permet de détecter des clusters de différentes tailles et formes, même si les données sont bruyantes ou si les clusters ont des densités variées. Il se distingue des autres algorithmes de clustering en utilisant la densité locale des points plutôt que des métriques de distance.

2.3 Perceptron multi-couches (MLP) :

Un Multi-Layer Perceptron (MLP) est un réseaux de neurones avec une couche d'entrée, un couches de sortie et une ou plusieurs couches intermédiaires. Chaque couche contient des neurones qui sont tous connectés avec les neurones de couche voisine et ont un fonction d'activation (par exemple : relu, sigmoid,etc...). Les paramètres d'un MLP sont : le nombre de couches intermédiaires, le nombre de neurones dans chaque couche, la fonction d'activation, le taux d'apprentissage, la fonction d'erreur et les poids. Les données qui seront utilisés pour notre réseau de neurones sont tous numériques, donc on est pas besoin de faire le recodage. Par ailleurs, la couche d'entrée contient 16 neurones de sorte que chaque neurone représente un attribut (qu'on les a précisé avant) et la couche de sortie contient un seul neurone qui a deux valeurs : 0 si l'URL est non malveillante et 1 sinon. Concernant les couches intermédiaires, le nombre de neurones et la fonction d'activation dans chacune d'elles seront modifiés à chaque fois afin de déduire leurs influenceances sur la précision du modèle.

Chapitre 3

Résultats expérimentaux

Knn : Pour l'évaluation de l'algorithme knn, on va calculer l'accuracy et la matrice de confusion et le taux d'erreur. Le score de cet algorithme est 96% avec un taux d'erreur de 0.03, donc notre algo à réussir de classifier les données. La figure suivante montre le rapport de classification : la precision, recall, f1-score et support.

Test Accuracy: 96.55524765231492				
Classification Report:				
	precision	recall	f1-score	support
benign	0.97	0.99	0.98	138296
malicious	0.97	0.88	0.92	41775
accuracy			0.97	180071
macro avg	0.97	0.94	0.95	180071
weighted avg	0.97	0.97	0.96	180071

FIGURE 3.1 – le rapport de classification

Naive Bayes : Le paramètre principal de cet algorithme est le var_smoothing. Il concerne le poids donné aux valeurs plus distantes de la moyenne de la distribution (gaussienne), et visuellement il a relation avec le lissage de la courbe de la distribution.

Nous nous focaliserons sur l'accuracy du modèle pour des valeurs discrètes de var_smoothing dans l'ensemble suivant : 2e-9 ; 2.5e-9 ; 3e-9 ; 3.5e-9 ; 4e-9 ; 4.5e-9

2e-9	2.5e-9	3e-9	3.5e-9	4e-9	4.5e-9
0.99145337	0.99144226	0.99143671	0.99132564	0.99131453	0.99131453

TABLE 3.1 – Valeur de l'accuracy selon le var_smoothing

Ensuite, nous envisageons d'automatiquement savoir la valeur du var_smoothing qui offre la meilleure accuracy, ce qui nous donne une valeur maximale de 0.99153667, pour un var_smoothing proche de 2e-9.

Cette valeur reste maximale lorsqu'on utilise un split aléatoire, ou lorsqu'on utilise un cv de 10 (au lieu de 5).

Arbre de décision : Voici les résultats trouvées pour l'exécution de l'algorithme de l'arbre de décision en laissant tous les paramètres par défaut :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	138130
1	0.99	0.99	0.99	41941
accuracy			1.00	180071
macro avg	0.99	0.99	0.99	180071
weighted avg	1.00	1.00	1.00	180071

FIGURE 3.2 – Mesures de performances pour l’arbre de décision

SVM : On rencontre une accuracy de 99.1%.

```
# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.8f}%".format(accuracy * 100))

, Accuracy: 99.16643990%
```

FIGURE 3.3 – Accuracy pour SVM

Clustering hiérarchique : L’application du Clustering Hiérarchique pour notre problématique a fait apparaître la première erreur qui nous est parvenue pour l’apprentissage non-supervisé : la grande taille de notre dataset a élevé la complexité de l’algorithme à des niveaux que Google Colab [?] n’a pas pu accomoder, et que les specifications de nos ordinateurs portables n’ont pas pu finir en temps raisonnable.

Nous avons essayés plusieurs méthodes pour réduire la complexité, et l’usage trop élevé de la RAM :

1. L’élimination d’attributs en utilisant la matrice de corrélation, le PCA, et même la réduction arbitraire du nombre d’attributs en 4, mais ça reste toujours trop demandant au niveau de la RAM.
2. L’augmentation du niveau d’erreur acceptable à des valeurs bien exagérées, mais il paraît que l’algorithme échoue à une étape critique directement liée au nombre d’enregistrement dans le dataset.
3. Changement de la distance (link average, etc) utilisée.

K-means : Pour une bonne application de cette algorithme, il faut bien choisir le nombre K, pour cela, on a utiliser la methode de coude(La méthode Elbow) qui va nous aider à trouver le k optimale pour la classification des cluster. Dans notre cas et d’après le graphe, on a trouver le k=2.

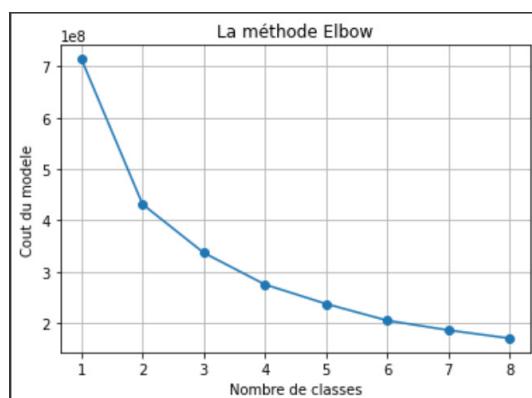


FIGURE 3.4 – Elbow method

On va passer maintenant à l'étape de l'apprentissage qui est bien detailler dans l'annexe (partie k-means).

Il existe plusieurs façons pour évaluer la performance d'un algorithme K-means. On utilise ici des mesures de validation internes (basées sur le clustering lui-même) et externes (basées sur les vrais labels). Les mesures internes utilisées sont :

Score de silhouette (0.52) : mesure de la similitude d'un objet avec son propre cluster en le comparant à d'autres clusters. Indique que les points sont bien classés.

Indice Davies-Bouldin (0.91) : mesure de la similarité entre chaque cluster et son cluster le plus similaire. Indique que les grappes sont plus distinctes et bien séparées.

Indice Calinski-Harabasz (295074) : mesure du rapport entre la variance entre les grappes et la variance entre les grappes. Indique un meilleur clustering et des grappes plus distinctes.

Les mesures externes utilisées sont :

Index Rand Ajusté (0.01) : mesure de la similarité entre les vraies étiquettes et les étiquettes prédites. Indique un mauvais model.

Indice Fowlkes-Mallows (0.65) : mesure de la moyenne géométrique de la précision et du rappel entre les étiquettes réelles et les étiquettes prévues. Indique que les vraies étiquettes et les étiquettes prévues sont un peu identiques.

On a trouvé aussi le taux d'erreur égal à 1 donc le modèle n'est pas en mesure de regrouper efficacement les données et que la somme des distances au carré entre les points et leur centroïde le plus proche est à sa valeur maximale possible, qui est 1. Cela pourrait être à cause de mauvaise performance du modèle, mauvais choix du nombre de grappes, mauvaise initialisation des centroïdes ou mauvaise qualité des données d'entrée.

DBSCAN : Dû à la taille large de données, nous n'avons pas pu compléter plusieurs executions (ni les refaire dans le fichier principale de google colab sur lequel on travailler pour réunion nos efforts), dont l'une est le DBSCAN.

Nous trouvons la valeur du score silhouette de -0.149663, ce qui dénote qu'une valeur de epsilon de 0.1 présente une mauvaise configuration pour ce genre de problème, et le modèle nous donne des clusters mal formés, où les points dans les clusters ne sont pas trop similaires.

MLP : On a essayé de faire varier quelques paramètres de notre modèle pour déduire leurs influences sur l'accuracy, on distingue 4 cas :

1. Le premier cas : (c'est notre modèle de départ)
 - 2 couches intermédiaires (la 1ère avec 16 neurones et la 2ème avec 8 neurones)
 - les fonctions d'activation sont resp. relu, relu, relu, sigmoid
 - le taux d'apprentissage 0.0001
 - l'optimiseur Adam
 - fonction de perte binary_crossentropy
2. Le deuxième cas : (on a changé le nombre de neurone dans la première couche intermédiaire)
 - 2 couches intermédiaires (la 1ère avec 32 neurones et la 2ème avec 16 neurones)
 - les fonctions d'activation sont resp. relu, relu, relu, sigmoid
 - le taux d'apprentissage 0.0001

- l'optimiseur Adam
 - fonction de perte binary_crossentropy
3. Le troisième cas : (on a changé les fonctions d'activation)
- 2 couches intermédiaires (la 1ère avec 16 neurones et la 2ème avec 8 neurones)
 - les fonctions d'activation sont resp. sigmoid, sigmoid, sigmoid, relu
 - le taux d'apprentissage 0.0001
 - l'optimiseur Adam
 - fonction de perte binary_crossentropy
4. Le quatrième cas : (on a ajouté 2 autres couches intermédiaires)
- 4 couches intermédiaires (avec resp. 16, 8, 64, 32 neurones)
 - les fonctions d'activation sont resp. relu, relu, relu, relu, sigmoid, sigmoid
 - le taux d'apprentissage 0.0001
 - l'optimiseur Adam
 - fonction de perte binary_crossentropy

Et la précision dans chaque cas vaut :

cas	loss	acc
1	0,015523	0,996702
2	0,016262	0,996724
3	0,043009	0,992270
4	0,014919	0,996811

TABLE 3.2 – Valeur de loss et acc selon le cas

Chapitre 4

Discussion

Dans cette partie, nous allons entamer une comparaison entre les différents résultats des algorithmes

4.1 Apprentissage Supervisé

Le KNN est atteint 96% pour son score, ce qui permet de juger de la non-existance du overfitting, ainsi que une bonne précision en ce qui concerne les données de URLs généralisés. Par ailleurs, le NB a une accuracy maximale de 0.991536, or ça peut être signe de over-fitting, donc il est conseillé de travailler avec une valeur de var_smoothing différente de 2e-9, par exemple 7e-9 ou même 1e-8, pour éviter les valeurs proches de 0.99. On confirme alors qu'un plus grand lissage permet de prendre en compte les points lointains. Par ailleurs, l'arbre de décision rencontre les mêmes problèmes de overfitting (99%), qui pourraient être évités en faisant un changement de paramètres pour changer la condition d'arrêt, par exemple la profondeur maximale permise. De même pour le SVM, cependant la grande taille des données n'a pas permis de faire multiple itérations sur l'algorithme pour juger l'influence des différents paramètres.

On trouve finalement que les algorithmes d'apprentissage supervisé ont une très grande précision pour ce genre de problème, qui est de classification pour deux classes.

4.2 Apprentissage Non-Supervisé

En ce qui concerne le non-supervisé, le K-Means nous donnent des mesures de performances très variées, jugeant les différents aspects du modèle. Il est très performant pour les variations internes, mais échoue au niveau des évaluations externes. Quant au MLP, on trouve de façon intéressante que l'accuracy ne change pas drastiquement, quelques soit les différentes configurations des paramètres. Le MLP reste robuste, et on atteint un maximum d'accuracy pour le quatrième cas, qui est d'ailleurs le plus différent de celui initial, et le plus détaillé. Cela montre que le MLP est une bonne approche en général, et une bonne base pour un algorithme assez précis. Par ailleurs, le DBSCAN et le Clustering hiérarchique sont trop complexe de temps et de RAM respectivement, ce qui implique le mal adaptation à ce genre de dataset, et un mauvais contexte pour leur application.

Conclusion

En conclusion, ce projet a démontré l'efficacité des algorithmes d'apprentissage supervisé et non-supervisé pour détecter les URL malveillantes. Les algorithmes KNN et Naive Bayes ont obtenu des scores élevés pour la précision, mais ont également été sujets à l'overfitting. Les algorithmes d'arbre de décision et de SVM ont également montré des résultats similaires, mais n'ont pas pu être testés en profondeur en raison de la grande taille des données. Le K-Means a été performant pour les variations internes, mais a échoué pour les évaluations externes. Le MLP s'est avéré être une approche robuste et précise. Le DBSCAN et le Clustering hiérarchique ont été trop complexes en termes de temps et de RAM pour être adaptés à ce type de jeu de données. Le choix de la taille de jeu de données a également été un facteur limitant pour nos efforts. Il reste toujours vrai que les techniques d'apprentissage automatique peuvent protéger les utilisateurs contre les attaques de phishing et les intrusions informatiques en utilisant les caractéristiques de structure de l'URL et les informations de contexte. Il existe de nombreuses autres applications possibles pour l'apprentissage automatique dans ce domaine, comme l'utilisation d'algorithmes de réseaux de neurones pour améliorer la précision de la détection des URL malveillantes comme le CNN. Il y a encore beaucoup de potentiel pour améliorer les techniques de détection des URL malveillantes à l'aide de l'apprentissage automatique.

Annexes

Code Complet avec Commentaires

```
Importation de données

[ ] import numpy as np
      import pandas as pd

      import matplotlib.pyplot as plt
      import seaborn as sns

      import os

[ ] from google.colab import files
data_to_load = files.upload()

import io
df = pd.read_csv(io.BytesIO(data_to_load['urldata.csv']))

Choose Files No file chosen Upload widget is only available when the
Saving urldata.csv to urldata.csv
```

FIGURE 4.1 – Image 1

```
[ ] df.head() #les premiers éléments du tableau de données
```

	Unnamed: 0	url	label	result
0	0	https://www.google.com	benign	0
1	1	https://www.youtube.com	benign	0
2	2	https://www.facebook.com	benign	0
3	3	https://www.baidu.com	benign	0
4	4	https://www.wikipedia.org	benign	0


```
[ ] #on supprime les attributs qui n'apportent pas d'infos nécessaires
df = df.drop('Unnamed: 0',axis=1)
```



```
[ ] #le nombre d'exemples et le nombre d'attributs
df.shape
```



```
(450176, 3)
```

FIGURE 4.2 – Image 2

```
[ ] #les informations sur notre dataframe
df.info()
```

#	Column	Non-Null Count	Dtype
0	url	450176	non-null object
1	label	450176	non-null object
2	result	450176	non-null int64


```
[ ] #tester s'il y a de valeurs manquantes
df.isnull().sum()
```



```
url      0
label    0
result   0
dtype: int64
```


cela montre qu'il n'y a pas de valeurs manquantes

FIGURE 4.3 – Image 3

1 - DATA PREPROCESSING

On va extraire de chaque URL les attributs suivants :

En premier lieu, (des entiers)

- la longueur d'URL
- la longueur du Hostname
- la longueur du path
- la longueur du First Directory

FIGURE 4.4 – Image 4

En deuxième lieu, (des entiers)

- nombre de ':'
- nombre de '@'
- nombre de '?'
- nombre de '='
- nombre de '%'
- nombre de '='
- nombre de 'http'
- nombre de chiffres
- nombre de Lettres
- nombre de Directories

En troisième lieu, (attributs binaires)

- l'utilisation ou non de IP
- l'utilisation ou non de Shortening URL

FIGURE 4.5 – Image 5

1/ le premier groupe d'attributs

```
[ ] !pip install tld

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tld
  Downloading tld-0.12.6-py38-none-any.whl (412 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 412.2/412.2 KB 15.2 MB/s eta 0:00:00
Installing collected packages: tld
Successfully installed tld-0.12.6

[ ] from urllib.parse import urlparse
from tld import get_tld
import os.path

[ ] #longueur d'URL
df['longueur_url'] = df['url'].apply(lambda i: len(str(i)))

[ ] #la longueur du Hostname
df['longueur_hostname'] = df['url'].apply(lambda i: len(urlparse(i).netloc))
```

FIGURE 4.6 – Image 6

```
▶ #la longueur du path
df['longueur_path'] = df['url'].apply(lambda i: len(urlparse(i).path))

[ ] #la longueur du First Directory
def fd_length(url):
    urlpath= urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0

df['longueur_fd'] = df['url'].apply(lambda i: fd_length(i))

[ ] #la longueur du Top Level Domain
df['tld'] = df['url'].apply(lambda i: get_tld(i,fail_silently=True))
def tld_length(tld):
    try:
        return len(tld)
    except:
        return -1

df['longueur_tld'] = df['tld'].apply(lambda i: tld_length(i))
```

FIGURE 4.7 – Image 7

```
[ ] #revoir les premiers lignes de données après l'extraction des nouveaux attributs
df.head()

   url  label  result  longueur_url  longueur_hostname  longueur_path  longueur_fd  tld  longueur_tld
0  https://www.google.com  benign      0            22             14            0          0 .com         3
1  https://www.youtube.com  benign      0            23             15            0          0 .com         3
2  https://www.facebook.com  benign      0            24             16            0          0 .com         3
3  https://www.baidu.com  benign      0            21             13            0          0 .com         3
4  https://www.wikipedia.org  benign      0            25             17            0          0 .org         3

[ ] df = df.drop("tld",1)

```

FIGURE 4.8 – Image 8

	url	label	result	longueur_url	longueur_hostname	longueur_path	longueur_fd	longueur_tld
0	https://www.google.com	benign	0	22	14	0	0	3
1	https://www.youtube.com	benign	0	23	15	0	0	3
2	https://www.facebook.com	benign	0	24	16	0	0	3
3	https://www.baidu.com	benign	0	21	13	0	0	3
4	https://www.wikipedia.org	benign	0	25	17	0	0	3

2/ le deuxième groupe d'attributs

FIGURE 4.9 – Image 9

2/ le deuxième groupe d'attributs

```
[ ] #nombre de '-'
df['nb-'] = df['url'].apply(lambda i: i.count('-'))

[ ] #nombre de '@'
df['nb@'] = df['url'].apply(lambda i: i.count('@'))

[ ] #nombre de '?'
df['nb?'] = df['url'].apply(lambda i: i.count('?'))

[ ] #nombre de '.'
df['nb.'] = df['url'].apply(lambda i: i.count('.'))

[ ] #nombre de '%'
df['nb%'] = df['url'].apply(lambda i: i.count('%'))

[ ] #nombre de '='
df['nb='] = df['url'].apply(lambda i: i.count('='))
```

FIGURE 4.10 – Image 10

```
[ ] #nombre de 'http'
df['nb-http'] = df['url'].apply(lambda i : i.count('http'))
#nombre de 'https'
df['nb-https'] = df['url'].apply(lambda i : i.count('https'))

[ ] #nombre de 'www'
df['nb-www'] = df['url'].apply(lambda i: i.count('www'))

[ ] #nombre de chiffres
def chiffres_count(url):
    digits = 0
    for i in url:
        if i.isnumeric():
            digits = digits + 1
    return digits
df['nb-chiffres']= df['url'].apply(lambda i: chiffres_count(i))
```

FIGURE 4.11 – Image 11

```

❸ #nombre de lettres
def lettres_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters
df['nb_letters'] = df['url'].apply(lambda i: lettres_count(i))

[ ] #nombre de Directories
def nb_dir(url):
    urlDir = urlparse(url).path
    return urlDir.count('/')
df['nb_dir'] = df['url'].apply(lambda i: nb_dir(i))

[ ] #données après l'ajout de ces attributs
df.head()

```

	url	label	result	longueur_url	longueur_hostname	longueur_path	longueur_fd	longueur_tld	nb-	nb@	nb?	nb.	nb%	nb%	nb- http	nb- https	nb- www	nb- chiffres	nb- lettres
0	https://www.google.com	benign	0	22	14	0	0	3	0	0	0	2	0	0	1	1	1	0	
1	https://www.youtube.com	benign	0	23	15	0	0	3	0	0	0	2	0	0	1	1	1	0	
2	https://www.facebook.com	benign	0	24	16	0	0	3	0	0	0	2	0	0	1	1	1	0	
3	https://www.baidu.com	benign	0	21	13	0	0	3	0	0	0	2	0	0	1	1	1	0	
4	https://www.wikipedia.org	benign	0	25	17	0	0	3	0	0	0	2	0	0	1	1	1	0	

FIGURE 4.12 – Image 12

FIGURE 4.13 – Image 13

```

# l'utilisation ou non de shortening URL (c.à.d si on a une URL réduite)
def shortening_service(url):
    match = re.search('bit.ly|goo.gl|shorte.st|go2l.ink|x.co|ow.ly|t.co|tinyurl|tr.ly|is.gd|cli.gs|'
                      'yfrog.com|migre.me|ff.im|tiny.cc|url4.eu|twitt.ac|su.pr|twurl.nl|snipurl.com|'
                      'short.to|BudURL.com|ping.fm|post.ly|Just.as|bkite.com|snipr.com|fic.kr|loopt.us|'
                      'doiopt.com|short.ie|kl.am|wp.me|rubyurl.com|om.ly|to.ly|bit.do|t.co|lnkd.in|'
                      'db.tt|qr.ae|adf.ly|goo.gl|bitly.com|cur.lv|tinyurl.com|ow.ly|bit.ly|ity.i|m|'
                      'q.gs|is.gd|po.st|bc.vc|twithis.com|u.to|j.mp|buzurl.com|cutt.us|u.bb|yourls.org|'
                      'x.co|prettylinkpro.com|scrnch.me|filoops.info|vturl.com|qr.net|1ur1.com|tweez.me|v.gd|'
                      'tr.im|link.zip.net|', url)
    if match:
        return -1
    else:
        return 1
df[['short_url']] = df['url'].apply(lambda i: shortening_service(i))

[ ] #revoir les données après l'ajout de ces deux attributs
df.head()

```

FIGURE 4.14 – Image 14

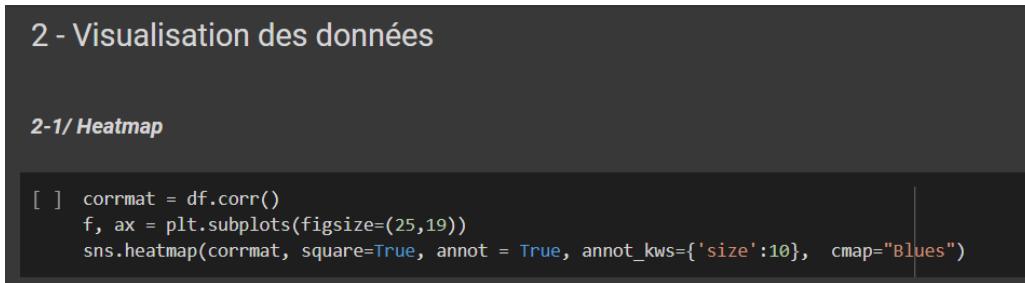


FIGURE 4.15 – Image 15

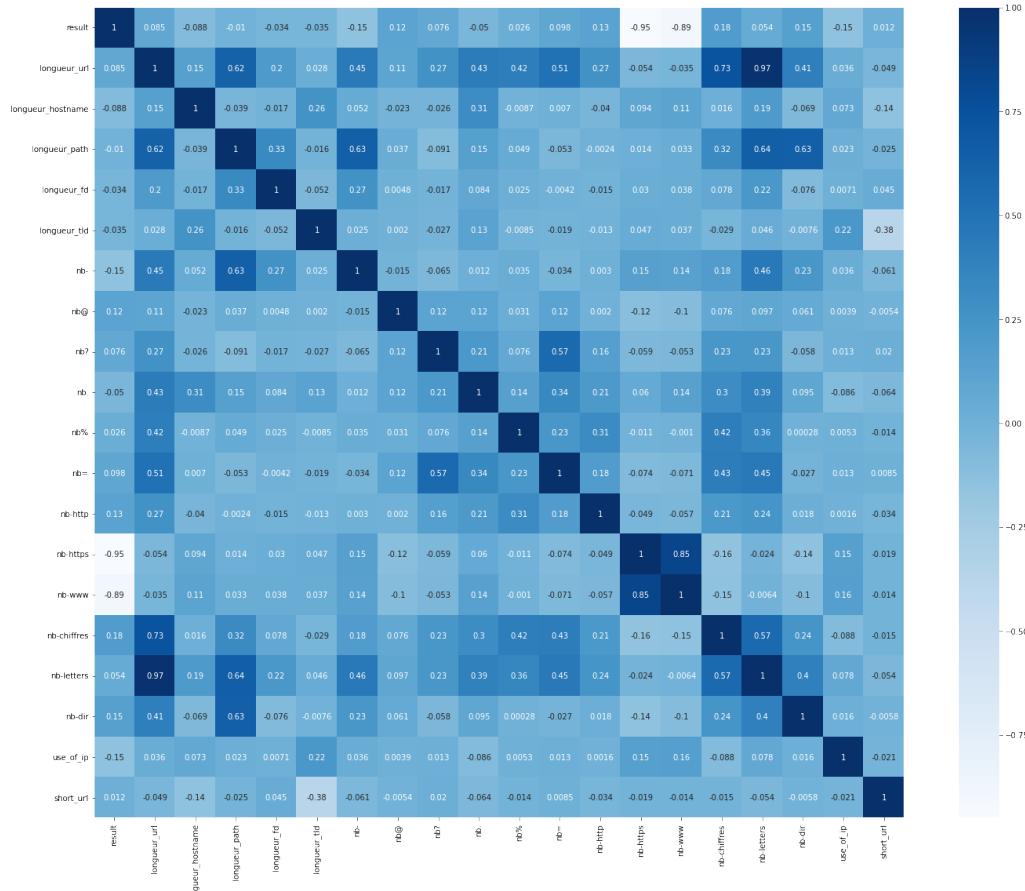


FIGURE 4.16 – Image 16

-2/ nombre d'URLs de chaque classe

```
[ ] plt.figure(figsize=(15,5))
sns.countplot(x='label',data=df)
plt.title("Nombre d'URLs par type",fontsize=20)
plt.xlabel("Type d'URLs",fontsize=18)
plt.ylabel("Nombre d'URLs",fontsize=18)
```

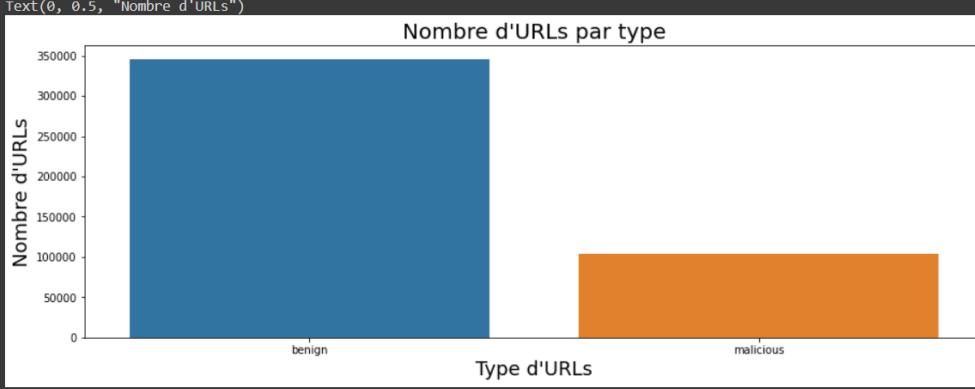


FIGURE 4.17 – Image 17

```
[ ] #pourcentage des URLs malveillantes et non malveillantes
print("Pourcentage des URLs malveillantes:{:.2f} {}".format(len(df[df['label']=='malicious'])/len(df['label'])*100))
print("Pourcentage des URLs non malveillantes:{:.2f} {}".format(len(df[df['label']=='benign'])/len(df['label'])*100))

Pourcentage des URLs malveillantes:23.20 %
Pourcentage des URLs non malveillantes:76.80 %
```

FIGURE 4.18 – Image 18

2-3/ Autres graphes

```
[ ] #longueurs d'URLs
plt.figure(figsize=(20,5))
plt.hist(df['longueur_url'],bins=50,color='LightBlue')
plt.title("longueur d'URL",fontsize=20)
plt.xlabel("longueur d'URL",fontsize=18)
plt.ylabel("Nombre d'URLs",fontsize=18)
plt.ylim(0,1000)
```

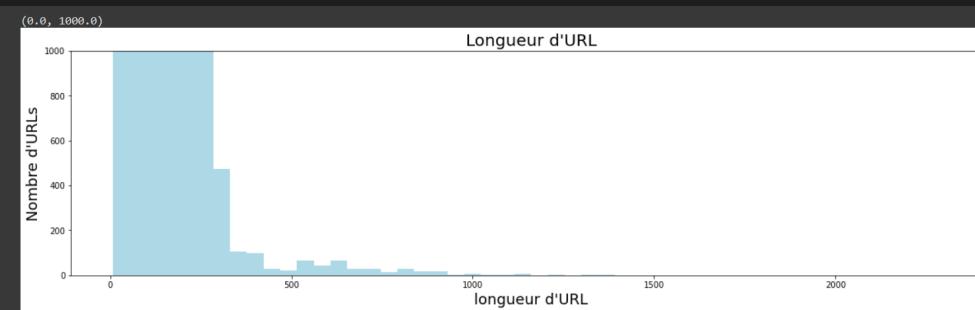


FIGURE 4.19 – Image 19

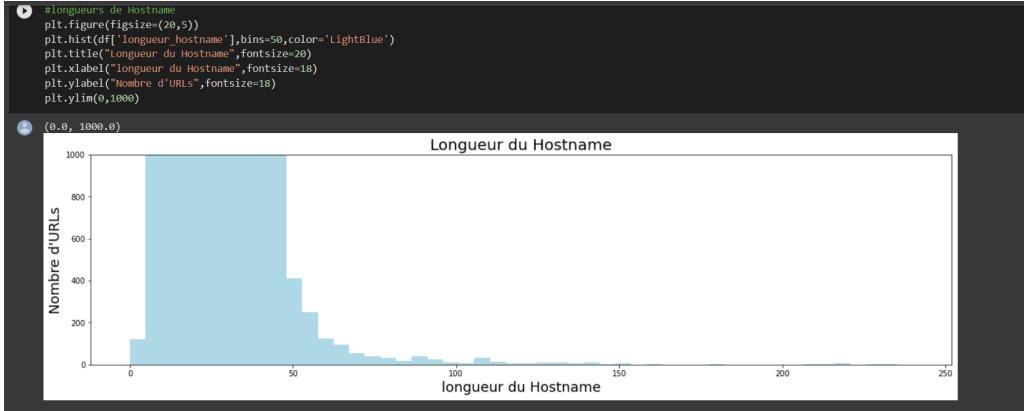


FIGURE 4.20 – Image 20

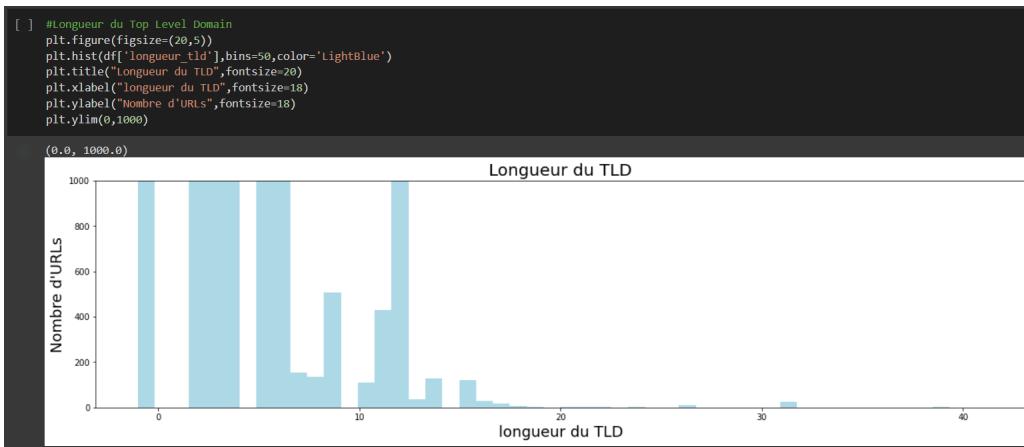


FIGURE 4.21 – Image 21

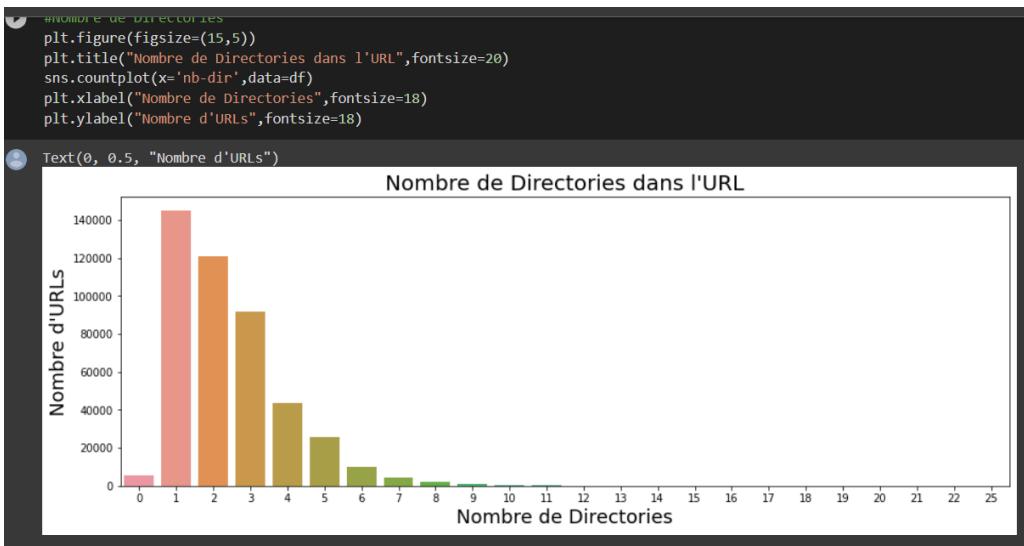


FIGURE 4.22 – Image 22

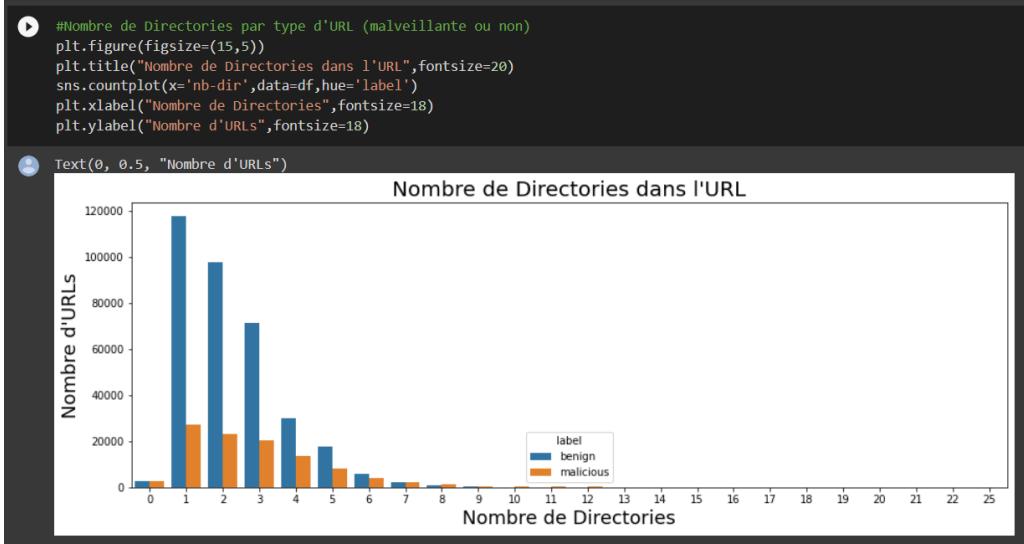


FIGURE 4.23 – Image 23

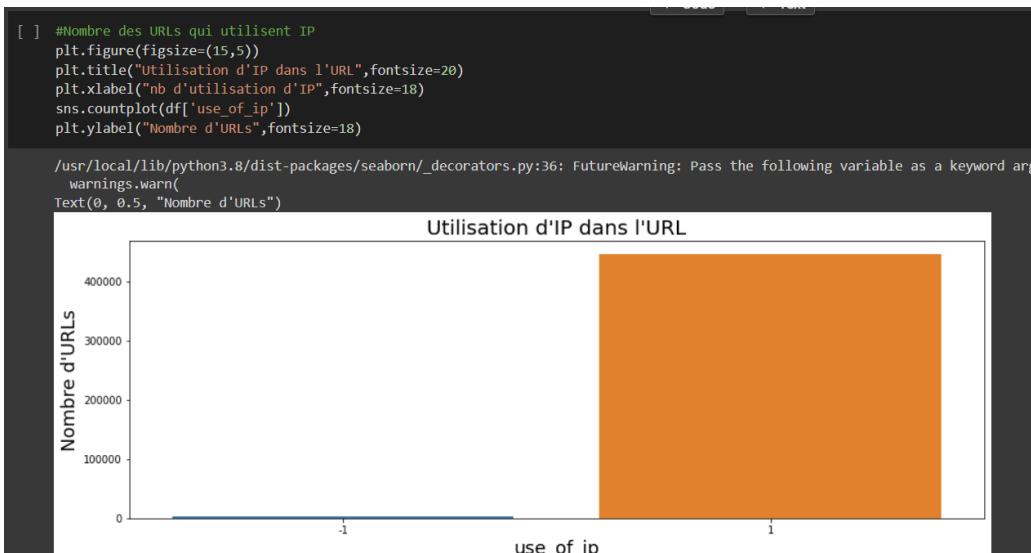


FIGURE 4.24 – Image 24

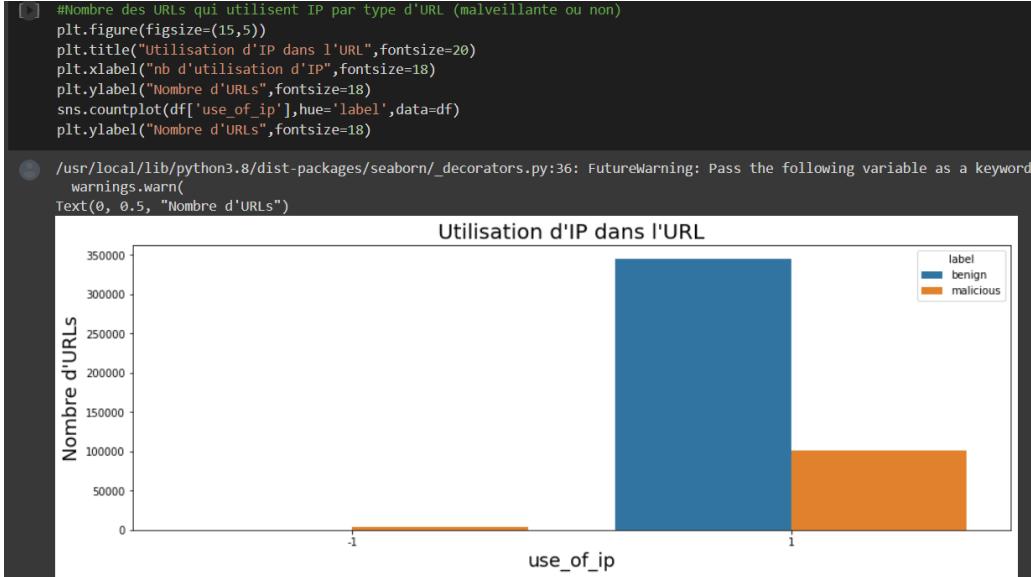


FIGURE 4.25 – Image 25

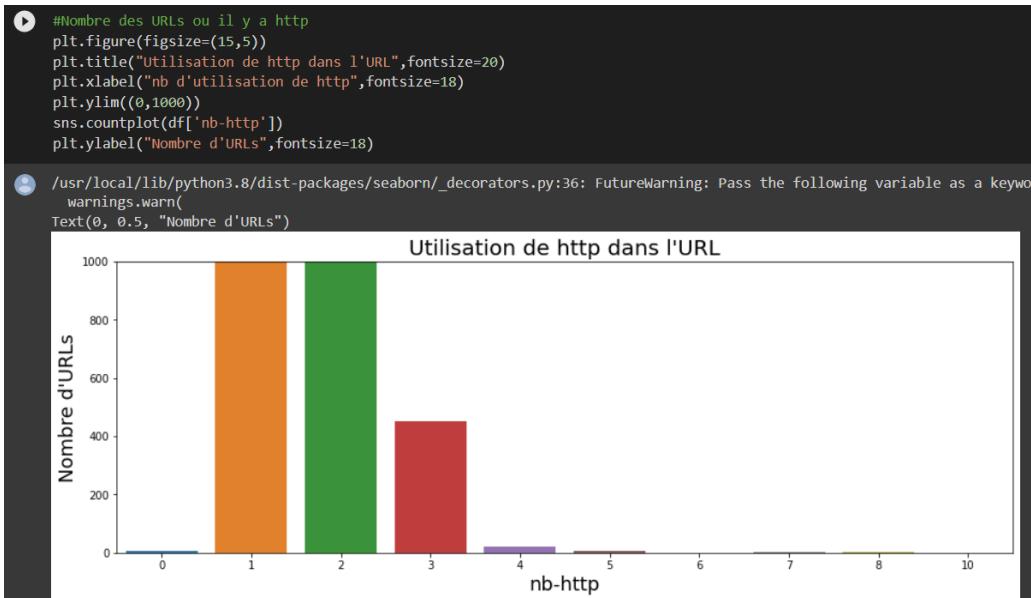


FIGURE 4.26 – Image 26



FIGURE 4.27 – Image 27

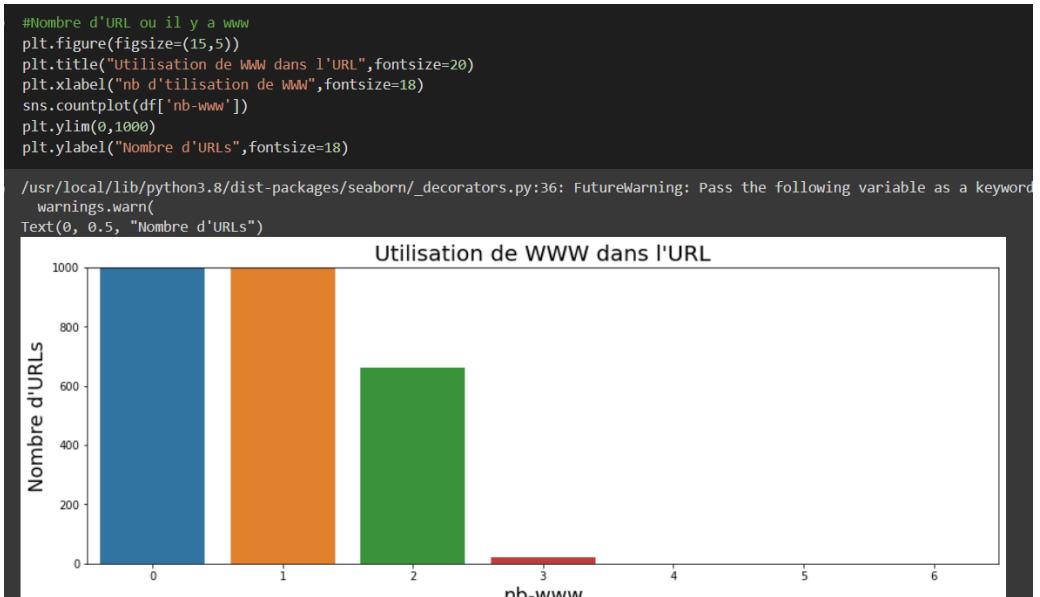


FIGURE 4.28 – Image 28

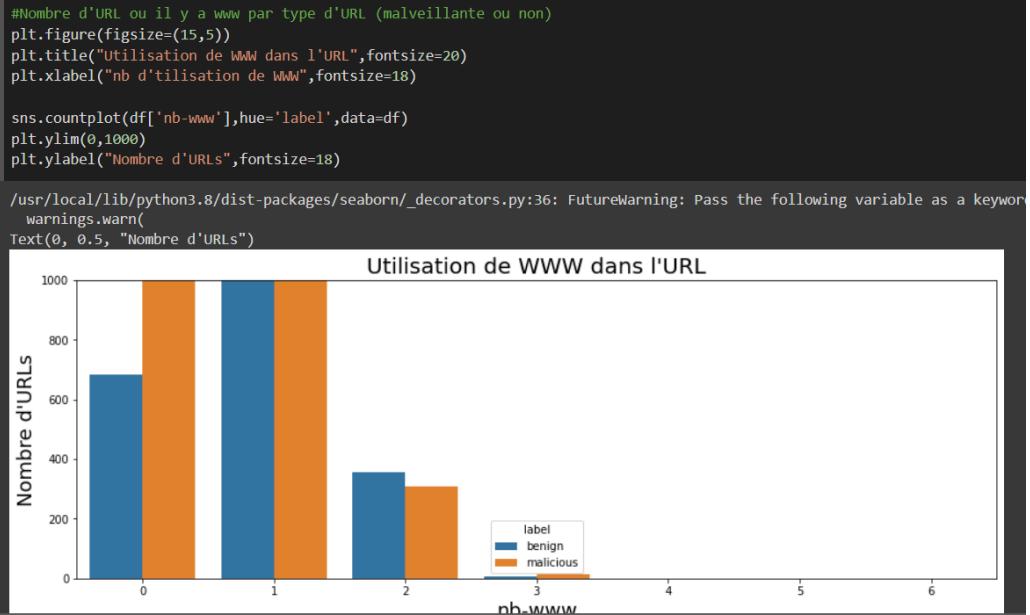


FIGURE 4.29 – Image 29

```
▶ df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 450176 entries, 0 to 450175
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   url              450176 non-null   object 
 1   label             450176 non-null   object 
 2   result            450176 non-null   int64  
 3   longueur_url     450176 non-null   int64  
 4   longueur_hostname 450176 non-null   int64  
 5   longueur_path    450176 non-null   int64  
 6   longueur_fd      450176 non-null   int64  
 7   longueur_tld     450176 non-null   int64  
 8   nb-               450176 non-null   int64  
 9   nb@               450176 non-null   int64  
 10  nb?               450176 non-null   int64  
 11  nb.               450176 non-null   int64  
 12  nb%               450176 non-null   int64  
 13  nb=               450176 non-null   int64  
 14  nb-http          450176 non-null   int64  
 15  nb-https         450176 non-null   int64  
 16  nb-www            450176 non-null   int64  
 17  nb-chiffres      450176 non-null   int64  
 18  nb-letters        450176 non-null   int64  
 19  nb-dir            450176 non-null   int64  
 20  use_of_ip         450176 non-null   int64  
 21  short_url         450176 non-null   int64  
dtypes: int64(20), object(2)
```

FIGURE 4.30 – Image 30

3 - Split des données

```
[ ] from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

[ ] #les variables indépendantes
x = df[['longueur_hostname',
        'longueur_path', 'longueur_fd', 'nb.', 'nb@', 'nb?',
        'nb%', 'nb.', 'nb=', 'nb-http', 'nb-https', 'nb-www', 'nb-chiffres',
        'nb-letters', 'nb-dir', 'use_of_ip']]

#variable cible
y = df['result']

[ ] !pip install imbalanced-learn

Requirement already satisfied: imbalanced-learn in c:\users\hp\anaconda3\lib\site-packages (0.10.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\hp\anaconda3\lib\site-packages (from imbalanced-learn) (2.2.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\hp\anaconda3\lib\site-packages (from imbalanced-learn) (1.7.3)
Requirement already satisfied: numpy>=1.17.3 in c:\users\hp\anaconda3\lib\site-packages (from imbalanced-learn) (1.21.5)
Requirement already satisfied: joblib>=1.1.1 in c:\users\hp\anaconda3\lib\site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: scikit-learn>=0.2 in c:\users\hp\anaconda3\lib\site-packages (from imbalanced-learn) (1.0.2)
```

FIGURE 4.31 – Image 31



#pour traiter le déséquilibre des données on utilise SMOTE

```
[ ] from imblearn import under_sampling, over_sampling

from imblearn.over_sampling import SMOTE

x_sample, y_sample = SMOTE().fit_resample(x, y.values.ravel())

x_sample = pd.DataFrame(x_sample)
y_sample = pd.DataFrame(y_sample)

# checking the sizes of the sample data
print("Size of x-sample :", x_sample.shape)
print("Size of y-sample :", y_sample.shape)

Size of x-sample : (691476, 16)
Size of y-sample : (691476, 1)
```




[] #données d'entraînement et données d'apprentissage

```
[ ] from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_sample, y_sample, test_size = 0.2) #20% test et 80% entraînement
print("Shape of x_train: ", x_train.shape)
print("Shape of x_valid: ", x_test.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of y_valid: ", y_test.shape)
```

FIGURE 4.32 – Image 32



Shape of x_train: (553180, 16)
 Shape of x_valid: (138296, 16)
 Shape of y_train: (553180, 1)
 Shape of y_valid: (138296, 1)

4 - knn

```
[ ] pip install keras

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: keras in /usr/local/lib/python3.8/dist-packages (2.9.0)

[ ] from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

[ ] from sys import version_info
#split to test and train data
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.6, stratify=y, random_state=42)

[ ] from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

FIGURE 4.33 – Image 33

```
[ ] #application de knn algo
knn = KNeighborsClassifier()
#apprentissage: # fit the model to our training data
knn.fit(x_train,y_train)
#test : # make predictions on the test set
pred=knn.predict(x_test)

#see if our modele is good : calculate the accuracy
score = knn.score(x_test,y_test)
print("score :",score,end="\n")

# taux d'erreur : calculate the error rate
error_rate = 1 - score
print("\n taux d'erreur :",error_rate)

score : 0.9559507083317136
taux d'erreur : 0.04404929166828642
```

FIGURE 4.34 – Image 34

```
▶ #evaluation :
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix

# calculate precision
precision = precision_score(y_test, pred)
print("Precision:", precision)

# calculate recall
recall = recall_score(y_test, pred)
print("Recall:", recall)

# calculate F1-score
f1 = f1_score(y_test, pred)
print("F1-score:", f1)

# calculate confusion matrix
conf_matrix = confusion_matrix(y_test, pred)
print("Confusion matrix:\n", conf_matrix)

#classification rapport
test_accuracy=list()
test_accuracy.append(score)

print("Test Accuracy: ", score*100, end="\n")
print("Classification Report: ")
print(classification_report(y_test, pred))

print("\n", test_accuracy)
```

FIGURE 4.35 – Image 35

```

Precision: 0.9552829122608766
Recall: 0.8499102333931777
F1-score: 0.8995211674393859
Confusion matrix:
[[136634  1662]
 [ 6270 35505]]
Test Accuracy: 95.59507083317136
Classification Report:
precision    recall    f1-score   support
          0       0.96      0.99      0.97     138296
          1       0.96      0.85      0.90      41775

accuracy                           0.96     180071
macro avg       0.96      0.92      0.94     180071
weighted avg    0.96      0.96      0.96     180071

[0.9559507083317136]

```

FIGURE 4.36 – Image 36

5 - Arbre de décision

Les paramètres sont :

- max_depth (la profondeur maximale de l'arbre, par défaut c'est non)
- min_samples_leaf (le nombre minimum d'échantillons requis pour se retrouver à un nœud de feuille, par défaut c'est 1)
- criterion (Représente la fonction de mesure de la qualité d'une scission. Il peut prendre soit le critère « gini » qui correspond à l'impureté de Gini soit « entropy» qui correspond au gain d'informations, par défaut c'est gini).

```

[ ] from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.tree import DecisionTreeClassifier, plot_tree

[ ] x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.6, random_state=42)
print("Shape of x_train: ", x_train.shape)
print("Shape of x_valid: ", x_test.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of y_valid: ", y_test.shape)

Shape of x_train: (270105, 16)
Shape of x_valid: (180071, 16)
Shape of y_train: (270105,)
Shape of y_valid: (180071,)

```

FIGURE 4.37 – Image 37

```

    dt_model = DecisionTreeClassifier() #instanciation

    dt_model.fit(x_train,y_train) #Apprentissage

DecisionTreeClassifier()

[ ] dt_predictions = dt_model.predict(x_test)
accuracy_score(y_test,dt_predictions) #calcul d'accuracy

0.9958238694737076

[ ] #la matrice de confusion
print(confusion_matrix(y_test,dt_predictions))

[[137774    356]
 [   396  41545]]

```

FIGURE 4.38 – Image 38

```

print(classification_report(y_test, dt_predictions))

precision    recall  f1-score   support

          0       1.00      1.00      1.00     138130
          1       0.99      0.99      0.99     41941

accuracy                           1.00     180071
macro avg       0.99      0.99      0.99     180071
weighted avg     1.00      1.00      1.00     180071


[ ] dt_predictions_tr = dt_model.predict(x_train)
print(classification_report(y_train, dt_predictions_tr))

precision    recall  f1-score   support

          0       1.00      1.00      1.00     207608
          1       1.00      1.00      1.00      62497

accuracy                           1.00     270105
macro avg       1.00      1.00      1.00     270105
weighted avg     1.00      1.00      1.00     270105

```

FIGURE 4.39 – Image 39

6 - Naive Bayes

```
[ ] from sklearn.naive_bayes import GaussianNB  
  
[ ] #train the model  
clf=GaussianNB()  
clf.fit(x_train,y_train)  
  
[ ] from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix  
  
#evaluate the model  
  
y_pred=clf.predict(x_test)  
  
# calculate precision  
precision = precision_score(y_test, pred,average="binary", pos_label='benign')  
print("Precision:", precision)  
  
# calculate recall  
recall = recall_score(y_test, pred,average="binary", pos_label='benign')  
print("Recall:", recall)  
  
# calculate F1-score  
f1 = f1_score(y_test, pred,average="binary", pos_label='benign')  
print("F1-score:", f1)
```

FIGURE 4.40 – Image 40

```
# calculate F1-score  
f1 = f1_score(y_test, pred,average="binary", pos_label='benign')  
print("F1-score:", f1)  
  
# calculate confusion matrix  
conf_matrix = confusion_matrix(y_test, pred)  
print("Confusion matrix:\n", conf_matrix)  
  
#classification rapport  
test_accuracy=list()  
test_accuracy.append(score)  
  
print("Test Accuracy: ", score*100, end="\n")  
print("Classification Report: ")  
print(classification_report(y_test, pred))  
  
print("\n", test_accuracy)
```

FIGURE 4.41 – Image 41

```
[ ] #puisque var_smoothing et de l'ordre de 1e-9 on peut essayer dans un premiere temps pour les valeurs 2e-9,2.5e-9,3e-9,3.5e-9,4e-9,4.5e-9
for i in [2e-9,2.5e-9,3e-9,3.5e-9,4e-9,4.5e-9]:
    gnb=GaussianNB(var_smoothing=i)
    #fit the model
    gnb.fit(x_train,y_train)
    #predicting
    y_pred=gnb.predict(x_test)
    #calculate accuracy
    accuracy=gnb.score(x_test, y_test)
    confusion_M=confusion_matrix(y_test,y_pred)
    #or
    #accuracy=accuracy_score(y_pred, y_test)
    print('accuracy de', i , '=', accuracy)

[ ] #on a une meilleur accuracy pour var_smoothing=2e-09
```

FIGURE 4.42 – Image 42

```
▶ #dans un deuxieme temps on vas essayer pour des valeur comprisent entre 2e-9,4e-9
#dans ce cas le model utilise un split de type cross_validation
#avec nombre de folders est cv=5
from sklearn.model_selection import GridSearchCV
nb_classifier = GaussianNB()

params_NB = {'var_smoothing': np.linspace(2e-9,4e-9,1000)}
gs_NB = GridSearchCV(estimator=nb_classifier,
                      param_grid=params_NB,
                      cv=5,    # use any cross validation technique
                      verbose=1,
                      scoring='accuracy')

gs_NB.fit(x, y)
print('le best parameter est',gs_NB.best_params_)
print('le best score est', gs_NB.best_score_)
print(classification_report(y_test,y_pred))

[ ] #on essay ce parametre avec le split aleatoire
Gauss_cls=GaussianNB(var_smoothing=3.203203203203203e-09)
Gauss_cls.fit(x_train,y_train)
y_pred=Gauss_cls.predict(x_test)
accuracy=Gauss_cls.score(x_test, y_test)
#confusion_M=confusion_matrix(y_test,y_pred)
print('accuracy: ', accuracy)
```

FIGURE 4.43 – Image 43

```

] # on utilise toujours un split de type cross_validation
#avec nombre de folders est cv=10
nb_classifier = GaussianNB()

params_NB = {'var_smoothing': np.linspace(2e-9,4e-9,1000)}
gs_NB = GridSearchCV(estimator=nb_classifier,
                      param_grid=params_NB,
                      cv=10,    # use any cross validation technique
                      verbose=1,
                      scoring='accuracy')
gs_NB.fit(x, y)
print('le best parameter est',gs_NB.best_params_)
print('le best score est', gs_NB.best_score_)
print(classification_report(y_test,y_pred))

] #on essay ce parametre avec le split aleatoire
Gauss_cls=GaussianNB(var_smoothing=3.203203203203203e-09)
Gauss_cls.fit(x_train,y_train)
y_pred=Gauss_cls.predict(x_test)
accuracy=Gauss_cls.score(x_test, y_test)
#confusion_M=confusion_matrix(y_test,y_pred)
print('accuracy: ', accuracy)

```

FIGURE 4.44 – Image 44

7 - SVM

```

[ ] #Import svm model
from sklearn.svm import SVC

[ ] # on cree un objet svc
svc = SVC()

# phase d'apprentissage
svc.fit(x_train, y_train)

#prediction de donnees de test
y_pred = svc.predict(x_test)

#Calcul du precision
precision = precision_score(y_test, pred,average="binary", pos_label='benign')
print("Precision:", precision)

#Calcul du rappel
recall = recall_score(y_test, pred,average="binary", pos_label='benign')
print("Recall:", recall)

```

FIGURE 4.45 – Image 45

```

#Calcul du F1-score
f1 = f1_score(y_test, pred, average="binary", pos_label='benign')
print("F1-score:", f1)

#Calcul de la matrice de confusion
conf_matrix = confusion_matrix(y_test, pred)
print("Confusion matrix:\n", conf_matrix)

#Classification rapport
test_accuracy=list()
test_accuracy.append(score)

print("Test Accuracy: ", score*100, end="\n")
print("Classification Report: ")
print(classification_report(y_test, pred))

print("\n", test_accuracy)

[ ] #tester sur l'argument kernel:
#kernel precise la forme des hyperplans de separation;
from concurrent.futures import ThreadPoolExecutor # pour utiliser multitreading afin d'augmenter la vitesse d'execution
from sklearn.model_selection import GridSearchCV # pour chercher les arguments les plus presis

```

FIGURE 4.46 – Image 46

```

param_grid = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}
svc = SVC()

# Utiliser 4 threads pour l'exécution
executor = ThreadPoolExecutor(max_workers=4)

# Utiliser les 4 workers(threads) pour exécuter gridsearch
future = executor.submit(GridSearchCV, svc, param_grid, scoring='accuracy', cv=5)

# Prendre les résultats
grid_search = future.result()
grid_search.fit(x, y)

# Afficher les meilleurs paramètres
print(grid_search.best_params_)

# Afficher l'accuracy_score pour chaque fitting
results = grid_search.cv_results_

for mts, params in zip(results['mean_test_score'], results['params']):
    print("Accuracy: {:.10f} - Parameters: {}".format(mts, params))
    print(classification_report(y_test, y_pred))

```

FIGURE 4.47 – Image 47

```

❶ # nous avons decide de travailler avec kernel='rbf' a cause de la difficulte de trouver une separation liniere entre les variables endogenes et car il a la meme precision que kernel='linear'. C est le parametre de penalite du terme d'erreur. Il controle le compromis entre la frontiere de decision et la classification correcte des points d'apprentissage.

param_grid = {'kernel': ['rbf'], 'C' : [40,80,160,300,500,1000]}
svc = SVC()

# utiliser 4 threads pour l'exécution
executor = ThreadPoolExecutor(max_workers=4)

# utiliser les 4 workers(threads) pour executer gridsearch
future = executor.submit(gridSearchCV, svc, param_grid, scoring='accuracy', cv=5)

# prendre les résultats
grid_search = future.result()
grid_search.fit(x, y)

# Print the best parameters
print(grid_search.best_params_)

[ ] # afficher l'accuracy_score pour chaque fitting
results = grid_search.cv_results_

for mts, params in zip(results['mean_test_score'], results['params']):
    print("Accuracy: {:.10f} - Parameters: {}".format(mts, params))
    print(classification_report(y_test,y_pred))

```

FIGURE 4.48 – Image 48

```

❶ #tester sur l'argument gamma :
#gamma est un paramètre pour les hyperplans non linéaires. Plus la valeur gamma est élevée, plus il essaie de s'adapter exactement à l'ensemble de données d'entraînement.
# nb : j'ai choisi à la fin de travailler avec gamma='scale' car les données ne sont pas centrées et réduites

param_grid = {'kernel': ['rbf'], 'C' : [40], 'gamma' : ['scale','auto',100,400,1000]}
svc = SVC()

# utiliser 4 threads pour l'exécution
executor = ThreadPoolExecutor(max_workers=4)

# utiliser les 4 workers(threads) pour executer gridsearch
future = executor.submit(gridSearchCV, svc, param_grid, scoring='accuracy', cv=5)

# prendre les résultats
grid_search = future.result()
grid_search.fit(x, y)

# Print the best parameters
print(grid_search.best_params_)

[ ] results = grid_search.cv_results_
for mts,params in zip(results['mean_test_score'], results['params']):
    print("Accuracy: {:.10f} - Parameters: {}".format(mts, params))

```

FIGURE 4.49 – Image 49

Test et résultats des 2 meilleurs modèles trouvés :

```

[ ] # creation d'object SVC
svc = SVC(kernel='rbf',C=40,gamma='scale')

# Fitting
svc.fit(x_train, y_train)

# predictions avec testing dataset
y_pred = svc.predict(x_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.8f}%".format(accuracy * 100))
print(classification_report(y_test,y_pred))

[ ] from sklearn.metrics import classification_report
# Print the precision and recall
print(classification_report(y_test, y_pred))

```

FIGURE 4.50 – Image 50

```

▶ # creation d'object svc
svc = SVC(gamma='scale',kernel='poly',degree=9,C=40)

# Fit the model to the training data
svc.fit(x_train, y_train)

# predictions avec testing dataset
y_pred = svc.predict(x_test)

# calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.8f}%".format(accuracy * 100))

[ ] from sklearn.metrics import classification_report
# Print the precision and recall
print(classification_report(y_test, y_pred))

```

FIGURE 4.51 – Image 51

8 - K-means

```

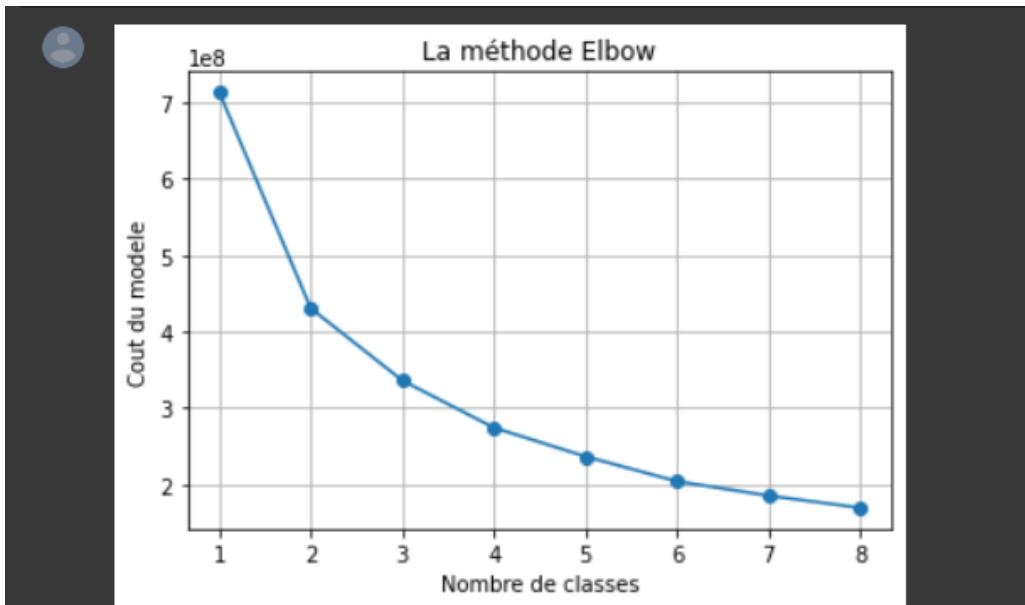
[ ] from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

[ ] #Détermination de la valeur optimale de K
liste= []
for i in range(1, 9):
    model= KMeans(n_clusters = i, init = 'k-means++', random_state = 30)
    model.fit(x)
    # Ajout de la somme des carrés à liste
    liste.append(model.inertia_)

[ ] # graphiquement
plt.plot(range(1, 9), liste, '-o')
plt.title('La méthode Elbow')
plt.xlabel('Nombre de classes')
plt.ylabel('cout du modèle')
plt.grid()
plt.show()

```

FIGURE 4.52 – Image 52



```
[ ] #methode de ELBOW : k=2
#Application de KMeans
model = KMeans(n_clusters=2)
model.fit(x)
y_pred=model.predict(x)
model_labels=model.labels_
model.inertia_
```

430254664.702217

FIGURE 4.53 – Image 53

```
# evaluation :
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score, adjusted_rand_score, normalized_mutual_info_score, fowlkes_mallows_score
#interne :
# calculate silhouette score
silhouette = silhouette_score(x, y_pred, metric = 'manhattan', sample_size=5000)
print("Silhouette score:", silhouette)

# calculate calinski harabasz score
calinski_harabasz = calinski_harabasz_score(x, y_pred)
print("Calinski Harabasz score:", calinski_harabasz)

# calculate the Davies-Bouldin index
db = davies_bouldin_score(x, y_pred)
print("Davies-Bouldin Index:", db)

#externe : test using the true value (true label)
# calculate the adjusted Rand index
ari = adjusted_rand_score(y, y_pred)
print("Adjusted Rand Index:", ari)

# calculate the normalized mutual information
nmi = normalized_mutual_info_score(y, y_pred)
print("Normalized Mutual Information:", nmi)
```

FIGURE 4.54 – Image 54

```

# calculate the fowlkes-mallows index
fmi = fowlkes_mallows_score(y, y_pred)
print("Fowlkes-Mallows Index:", fmi)

Silhouette score: 0.4967477911859196
Calinski harabasz score: 295873.7993602534
Davies-Bouldin Index: 0.9127923172686465
Adjusted Rand Index: 0.010502939300353703
Normalized Mutual Information: 0.00045000179918708484
Fowlkes-Mallows Index: 0.6555723964551047

#taux d'erreur
# initialize the error rate to 0
error_rate = 0

# iterate over each data point
for i in range(len(x)):
    # if the data point is not in the correct cluster, increment the error rate
    if y_pred[i] != y[i]:
        error_rate += 1

# calculate the error rate as a percentage
error_rate = error_rate / len(x)
print("Taux d'erreur :",error_rate)

Taux d'erreur : 0.340595678134774

```

FIGURE 4.55 – Image 55

```

▶ #essayer de corriger l'erreur
#la modification des parametres de k means
from sklearn.cluster import KMeans
import numpy as np

# Initialize the model with k=3
kmeans_1 = KMeans(n_clusters=2, init='random', random_state=0)
kmeans_2 = KMeans(n_clusters=2, init='k-means++', random_state=0)

# Fit the model on the data
kmeans_1.fit(x)
kmeans_2.fit(x)

# Get the cluster assignments for each point
clusters_1 = kmeans_1.predict(x)
clusters_2 = kmeans_2.predict(x)

# Get the coordinates of the cluster centers
centroids_1 = kmeans_1.cluster_centers_
centroids_2 = kmeans_2.cluster_centers_

# Calculate the within-cluster sum of squares
error_1 = kmeans_1.inertia_
error_2 = kmeans_2.inertia_
print("Error rate with random initialization:", error_1)
print("Error rate with k-means++ initialization:", error_2)

```

FIGURE 4.56 – Image 56

```
Error rate with random initialization: 430254264.38761747
Error rate with k-means++ initialization: 430254082.51903206
```

```
#essayer de trouver le bon k qui va donnees le bon score
list_score = []
for k in range(2, 9):
    model = KMeans(n_clusters = k).fit(x)
    labels = model.labels_
    list_score.append(silhouette_score(x, labels, metric = 'manhattan',sample_size=5000))

[ ] # Graghiquement
plt.plot(range(2, 9), list_score, '-o')
plt.xlabel('Nombre de classes')
plt.ylabel('Score')
plt.grid()
plt.show()
```

FIGURE 4.57 – Image 57

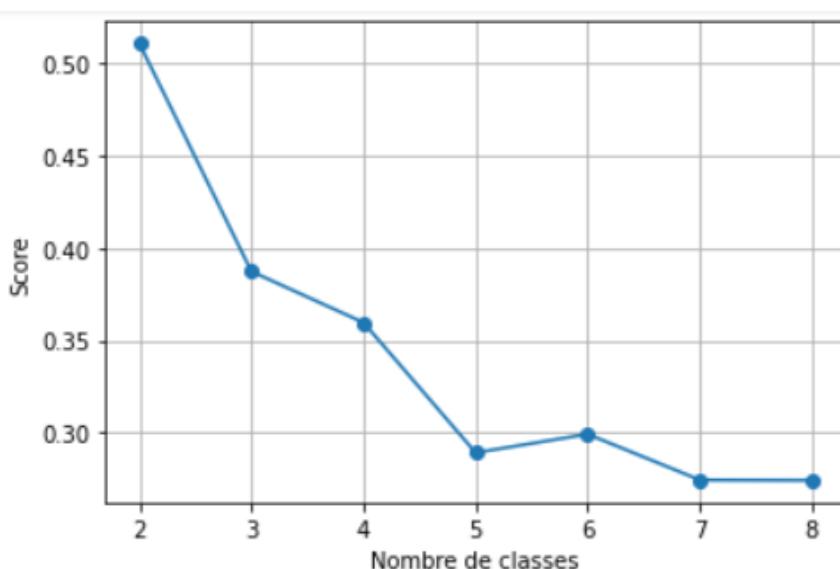


FIGURE 4.58 – Image 58

10 - DBSCAN

```
▶ from sklearn.cluster import DBSCAN
from sklearn import metrics

# choisir les paramètres de départ de DBSCAN
epsilon = 0.5
min_samples = 10

# réaliser le dbscan sur la data numérique, donc seulement les colonnes 3 à 19 inclus
data = df.iloc[:, 3:20]
```



```
[ ] from sklearn.metrics import silhouette_score
# Nous testons le score silhouette pour différentes valeurs de epsilon
eps_range = np.arange(0.1, 1, 0.1)
sil_scores = []
for eps in eps_range:
    db = DBSCAN(eps=eps, min_samples=min_samples)
    db.fit(data)
    labels = db.labels_
    sil_scores.append(silhouette_score(data, labels))
```

FIGURE 4.59 – Image 59



```
# en ignorant les données bruitantes, on retrouve le nombre de clusters pertinents
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

print("n_clusters = ", n_clusters)
```



```
[ ] # Visualisation
plt.plot(eps_range, sil_scores)
plt.xlabel("Epsilon")
plt.ylabel("Silhouette Score")
plt.show()
```



```
[ ]
```

FIGURE 4.60 – Image 60

11 - MLP

```
[ ] import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.callbacks import ModelCheckpoint
import keras
from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization ,Activation
from keras.utils import np_utils, to_categorical
from keras.callbacks import ModelCheckpoint
```

FIGURE 4.61 – Image 111

Construction du modèle

Les paramètres sont :

- le nombre de neurones
- la fonction d'activation
- le nombre de couches intermédiaires

Le modèle 1

```
[ ] #on commence par créer une instance du sequential()
model1 = Sequential()

#on définit maintenant les couches qu'on veut avoir dans notre modèle
model1.add(Dense(32, activation = 'relu', input_shape = (16, ))) #couche d'entrée
#des couches intermédiaires
model1.add(Dense(16, activation='relu'))
model1.add(Dense(8, activation='relu'))
model1.add(Dense(1, activation='sigmoid')) #couche de sortie
model1.summary()
```

FIGURE 4.62 – Image 112

Model: "sequential"		
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	544
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 8)	136
dense_3 (Dense)	(None, 1)	9
<hr/>		
Total params: 1,217		
Trainable params: 1,217		
Non-trainable params: 0		

Compilation du modèle
on choisit la fonction d'erreur (pour diminuer l'erreur)
l'optimiseur (pour obtenir des meilleures résultats pour la fonction de perte)
La métrique (pour évaluer le modèle)

FIGURE 4.63 – Image 113

```
❶ #l'optimiseur Adam est une méthode de descente de gradient stochastique qui prend en paramètre : le learning rate, beta1, beta2, epsilon, amsgrad
❷ #on a précisé juste le learning_rate en 0.0001 et on a laissé les autres paramètres par défaut
optimiseur1 = keras.optimizers.Adam(lr=0.0001)
❸ #la fonction de perte choisie ici est binary_crossentropy avec la métrique acc
model1.compile(optimizer= optimiseur1 ,loss='binary_crossentropy',metrics=['acc'])

❹ /usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
super(Adam, self).__init__(name, **kwargs)
```

FIGURE 4.64 – Image 114

Entrainement du modèle

```
[ ] checkpointer = ModelCheckpoint('url.h5', monitor='val_acc', mode='max', verbose=2, save_best_only=True)
history=model1.fit(x_train, y_train, batch_size=256, epochs=10, validation_data=(x_test, y_test), callbacks=[checkpointer])
#epochs : nombre d'itération
```

```
Epoch 1/10
2154/2161 [=====>.] - ETA: 0s - loss: 0.4918 - acc: 0.7990
Epoch 1: val_acc improved from -inf to 0.96124, saving model to url.h5
2161/2161 [=====>.] - 8s 3ms/step - loss: 0.4908 - acc: 0.7995 - val_loss: 0.1782 - val_acc: 0.9612
Epoch 2/10
2149/2161 [=====>.] - ETA: 0s - loss: 0.0835 - acc: 0.9829
Epoch 2: val_acc improved from 0.96124 to 0.99193, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0833 - acc: 0.9829 - val_loss: 0.0432 - val_acc: 0.9919
Epoch 3/10
2152/2161 [=====>.] - ETA: 0s - loss: 0.0305 - acc: 0.9949
Epoch 3: val_acc improved from 0.99193 to 0.99522, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0304 - acc: 0.9949 - val_loss: 0.0247 - val_acc: 0.9952
Epoch 4/10
2158/2161 [=====>.] - ETA: 0s - loss: 0.0210 - acc: 0.9962
Epoch 4: val_acc improved from 0.99522 to 0.99619, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0210 - acc: 0.9962 - val_loss: 0.0200 - val_acc: 0.9962
Epoch 5/10
2144/2161 [=====>.] - ETA: 0s - loss: 0.0185 - acc: 0.9964
Epoch 5: val_acc improved from 0.99619 to 0.99620, saving model to url.h5
2161/2161 [=====>.] - 6s 3ms/step - loss: 0.0185 - acc: 0.9964 - val_loss: 0.0187 - val_acc: 0.9962
Epoch 6/10
2150/2161 [=====>.] - ETA: 0s - loss: 0.0176 - acc: 0.9965
Epoch 6: val_acc improved from 0.99620 to 0.99649, saving model to url.h5
2161/2161 [=====>.] - 6s 3ms/step - loss: 0.0176 - acc: 0.9965 - val_loss: 0.0170 - val_acc: 0.9965
Epoch 7/10
2160/2161 [=====>.] - ETA: 0s - loss: 0.0169 - acc: 0.9966
Epoch 7: val_acc did not improve from 0.99649
2161/2161 [=====>.] - 8s 4ms/step - loss: 0.0169 - acc: 0.9966 - val_loss: 0.0165 - val_acc: 0.9965
Epoch 8/10
2142/2161 [=====>.] - ETA: 0s - loss: 0.0166 - acc: 0.9966
Epoch 8: val_acc did not improve from 0.99649
2161/2161 [=====>.] - 6s 3ms/step - loss: 0.0166 - acc: 0.9966 - val_loss: 0.0185 - val_acc: 0.9963
Epoch 9/10
2157/2161 [=====>.] - ETA: 0s - loss: 0.0162 - acc: 0.9967
Epoch 9: val_acc improved from 0.99649 to 0.99662, saving model to url.h5
2161/2161 [=====>.] - 8s 4ms/step - loss: 0.0163 - acc: 0.9967 - val_loss: 0.0161 - val_acc: 0.9966
Epoch 10/10
2149/2161 [=====>.] - ETA: 0s - loss: 0.0160 - acc: 0.9968
Epoch 10: val_acc improved from 0.99662 to 0.99670, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0160 - acc: 0.9968 - val_loss: 0.0155 - val_acc: 0.9967
```

FIGURE 4.65 – Image 115

```
Epoch 4/10
2158/2161 [=====>.] - ETA: 0s - loss: 0.0210 - acc: 0.9962
Epoch 4: val_acc improved from 0.99522 to 0.99619, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0210 - acc: 0.9962 - val_loss: 0.0200 - val_acc: 0.9962
Epoch 5/10
2144/2161 [=====>.] - ETA: 0s - loss: 0.0185 - acc: 0.9964
Epoch 5: val_acc improved from 0.99619 to 0.99620, saving model to url.h5
2161/2161 [=====>.] - 6s 3ms/step - loss: 0.0185 - acc: 0.9964 - val_loss: 0.0187 - val_acc: 0.9962
Epoch 6/10
2150/2161 [=====>.] - ETA: 0s - loss: 0.0176 - acc: 0.9965
Epoch 6: val_acc improved from 0.99620 to 0.99649, saving model to url.h5
2161/2161 [=====>.] - 6s 3ms/step - loss: 0.0176 - acc: 0.9965 - val_loss: 0.0170 - val_acc: 0.9965
Epoch 7/10
2160/2161 [=====>.] - ETA: 0s - loss: 0.0169 - acc: 0.9966
Epoch 7: val_acc did not improve from 0.99649
2161/2161 [=====>.] - 8s 4ms/step - loss: 0.0169 - acc: 0.9966 - val_loss: 0.0165 - val_acc: 0.9965
Epoch 8/10
2142/2161 [=====>.] - ETA: 0s - loss: 0.0166 - acc: 0.9966
Epoch 8: val_acc did not improve from 0.99649
2161/2161 [=====>.] - 6s 3ms/step - loss: 0.0166 - acc: 0.9966 - val_loss: 0.0185 - val_acc: 0.9963
Epoch 9/10
2157/2161 [=====>.] - ETA: 0s - loss: 0.0162 - acc: 0.9967
Epoch 9: val_acc improved from 0.99649 to 0.99662, saving model to url.h5
2161/2161 [=====>.] - 8s 4ms/step - loss: 0.0163 - acc: 0.9967 - val_loss: 0.0161 - val_acc: 0.9966
Epoch 10/10
2149/2161 [=====>.] - ETA: 0s - loss: 0.0160 - acc: 0.9968
Epoch 10: val_acc improved from 0.99662 to 0.99670, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0160 - acc: 0.9968 - val_loss: 0.0155 - val_acc: 0.9967
```

FIGURE 4.66 – Image 116

```
# la variation de Accuracy durant l'apprentissage
plt.plot(history.history['acc'], color='red')
plt.plot(history.history['val_acc'], color='green')
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train_acc', 'val_acc'], loc = 'upper right')
plt.show()
```

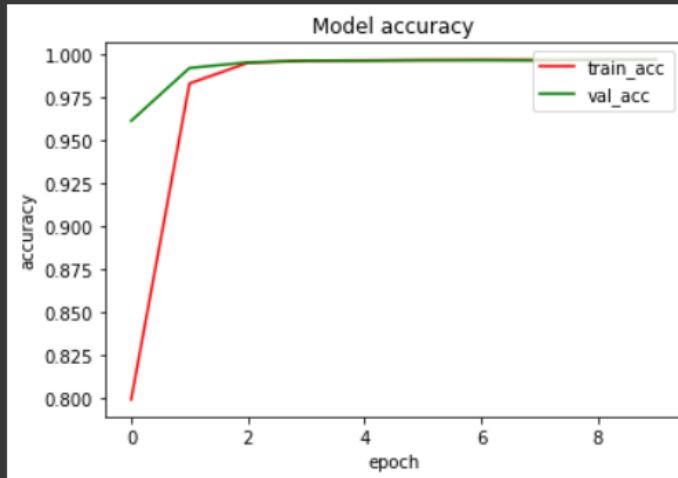


FIGURE 4.67 – Image 117

In Keras, `val_acc` is the accuracy of the model on the validation set, while `train_acc` is the accuracy of the model on the training set. The validation set is a portion of the data that is held out during training and is used to evaluate the performance of the model during training. The training set is used to fit the model's parameters. The difference between the training accuracy and validation accuracy can give an indication of how well the model is generalizing to new data, as a model with a high training accuracy but low validation accuracy is likely overfitting.

```
[ ] # la variation de loss durant l'apprentissage
plt.plot(history.history['loss'], color ='red')
plt.plot(history.history['val_loss'], color ='green')
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss', 'val_loss'], loc = 'upper right')
plt.show()
```

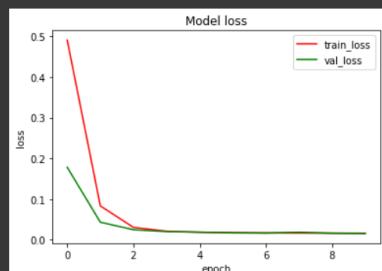


FIGURE 4.68 – Image 118

Evaluation du modèle

```
[ ] # predicting on test data.  
pred_test = model1.predict(x_test)  
for i in range (len(pred_test)):  
    if (pred_test[i] < 0.5):  
        pred_test[i] = 0  
    else:  
        pred_test[i] = 1  
pred_test = pred_test.astype(int)  
  
4322/4322 [=====] - 6s 1ms/step
```

```
[ ] def view_result(array):  
    array = np.array(array)  
    for i in range(len(array)):  
        if array[i] == 0:  
            print("Non malveillante")  
        else:  
            print("Malveillante")
```

FIGURE 4.69 – Image 119

```
[ ] view_result(pred_test[:10])
```

```
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante
```

```
[ ] view_result(y_test[:10])
```

```
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante
```

FIGURE 4.70 – Image 120

```

score = model1.evaluate( x_test, y_test, verbose = 0)
print(score)

[0.01552372332662344, 0.9967027306556702]

```

Le modèle 2

La différence par rapport au modèle 1 est qu'on va modifier le nombre de neurones dans les couches intermédiaires

```

[ ] model2 = Sequential()

model2.add(Dense(32, activation = 'relu', input_shape = (16, )))
model2.add(Dense(32, activation='relu'))
model2.add(Dense(16, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))
model2.summary()

Model: "sequential_1"

```

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 32)	544
dense_5 (Dense)	(None, 32)	1056
dense_6 (Dense)	(None, 16)	528
dense_7 (Dense)	(None, 1)	17

FIGURE 4.71 – Image 121

```

Total params: 2,145
Trainable params: 2,145
Non-trainable params: 0

[ ] optimiseur2= keras.optimizers.Adam(lr=0.0001)
model2.compile(optimizer= optimiseur2 ,loss='binary_crossentropy',metrics=['acc'])

[ ] checkpointer2 = ModelCheckpoint('url.h5', monitor='val_acc', mode='max', verbose=2, save_best_only=True)
history2=model2.fit(x_train, y_train, batch_size=256, epochs=5, validation_data=(x_test, y_test), callbacks=[checkpointer2])

Epoch 1/5
2159/2161 [=====>] - ETA: 0s - loss: 0.3620 - acc: 0.8845
Epoch 1: val.acc improved from -inf to 0.98027, saving model to url.h5
2161/2161 [=====] - 13s 5ms/step - loss: 0.3618 - acc: 0.8846 - val_loss: 0.0807 - val_acc: 0.9803
Epoch 2/5
2159/2161 [=====>] - ETA: 0s - loss: 0.0465 - acc: 0.9913
Epoch 2: val.acc improved from 0.98027 to 0.99514, saving model to url.h5
2161/2161 [=====] - 7s 3ms/step - loss: 0.0465 - acc: 0.9913 - val_loss: 0.0281 - val_acc: 0.9951
Epoch 3/5
2159/2161 [=====>] - ETA: 0s - loss: 0.0244 - acc: 0.9954
Epoch 3: val.acc improved from 0.99514 to 0.99627, saving model to url.h5
2161/2161 [=====] - 6s 3ms/step - loss: 0.0245 - acc: 0.9954 - val_loss: 0.0198 - val_acc: 0.9964
Epoch 4/5
2155/2161 [=====>] - ETA: 0s - loss: 0.0199 - acc: 0.9960
Epoch 4: val.acc improved from 0.99627 to 0.99636, saving model to url.h5
2161/2161 [=====] - 7s 3ms/step - loss: 0.0199 - acc: 0.9960 - val_loss: 0.0176 - val_acc: 0.9964
Epoch 5/5

```

FIGURE 4.72 – Image 122

```

2155/2161 [=====>.] - ETA: 0s - loss: 0.0199 - acc: 0.9960
Epoch 4: val_acc improved from 0.99627 to 0.99636, saving model to url.h5
2161/2161 [=====] - 7s 3ms/step - loss: 0.0199 - acc: 0.9960 - val_loss: 0.0176 - val_acc: 0.9964
Epoch 5/5
2147/2161 [=====>.] - ETA: 0s - loss: 0.0184 - acc: 0.9963
Epoch 5: val_acc improved from 0.99636 to 0.99672, saving model to url.h5
2161/2161 [=====] - 6s 3ms/step - loss: 0.0184 - acc: 0.9963 - val_loss: 0.0163 - val_acc: 0.9967

```

```

[ ] # la variation de Accuracy durant l'apprentissage
plt.plot(history2.history['acc'], color='red')
plt.plot(history2.history['val_acc'], color='green')
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train_acc','val_acc'], loc = 'upper right')
plt.show()

```

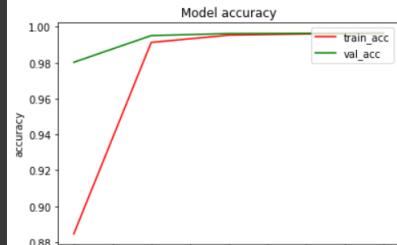


FIGURE 4.73 – Image 123

```

[ ] # la variation de loss durant l'apprentissage
plt.plot(history2.history['loss'], color ='red')
plt.plot(history2.history['val_loss'], color ='green')
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss','val_loss'], loc = 'upper right')
plt.show()

```

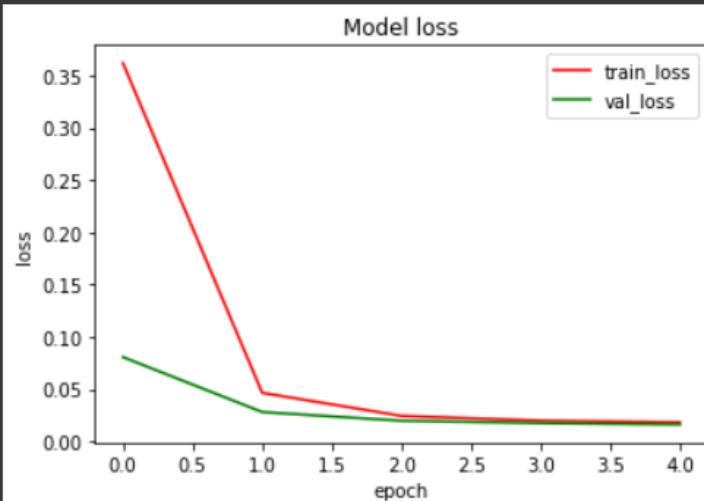


FIGURE 4.74 – Image 124

```
[ ] # predicting on test data.  
pred_test2 = model2.predict(x_test)  
for i in range (len(pred_test2)):  
    if (pred_test2[i] < 0.5):  
        pred_test2[i] = 0  
    else:  
        pred_test2[i] = 1  
pred_test2 = pred_test2.astype(int)  
  
4322/4322 [=====] - 6s 1ms/step  
  
[ ] view_result(pred_test2[:10])  
  
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante
```

FIGURE 4.75 – Image 125

```
▶ view_result(y_test[:10])  
  
👤 Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante  
  
[ ] score2 = model2.evaluate( x_test, y_test, verbose = 0)  
print(score2)  
  
[0.01626245118677616, 0.9967244267463684]  
  
Le modèle 3
```

FIGURE 4.76 – Image 126

Le modèle 3

On va changer les fonctions d'activation

```
[ ] model3 = Sequential()

model3.add(Dense(32, activation = 'sigmoid', input_shape = (16, )))
model3.add(Dense(16, activation='sigmoid'))
model3.add(Dense(8, activation='sigmoid'))
model3.add(Dense(1, activation='relu'))
model3.summary()

Model: "sequential_2"

Layer (type)          Output Shape         Param #
=====
dense_8 (Dense)      (None, 32)           544
dense_9 (Dense)      (None, 16)            528
dense_10 (Dense)     (None, 8)             136
dense_11 (Dense)     (None, 1)             9
=====
Total params: 1,217
Trainable params: 1,217
Non-trainable params: 0
```

FIGURE 4.77 – Image 127

```
[ ] optimiseur3 = Keras.optimizers.Adam(lr=0.0001)
model3.compile(optimizer= optimiseur3 ,loss='binary_crossentropy',metrics=['acc'])

/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use
super(Adam, self).__init__(name, **kwargs)

[ ] checkpointer3 = ModelCheckpoint('url.h5', monitor='val_acc', mode='max', verbose=2, save_best_only=True)
history3=model3.fit(x_train, y_train, batch_size=256, epochs=5, validation_data=(x_test, y_test), callbacks=[checkpointer3])

Epoch 1/5
2153/2161 [=====>.] - ETA: 0s - loss: 0.6070 - acc: 0.6878
Epoch 1: val_acc improved from -inf to 0.89450, saving model to url.h5
2161/2161 [=====] - 6s 3ms/step - loss: 0.6061 - acc: 0.6885 - val_loss: 0.3688 - val_acc: 0.8945
Epoch 2/5
2148/2161 [=====>.] - ETA: 0s - loss: 0.1782 - acc: 0.9465
Epoch 2: val_acc improved from 0.89450 to 0.97661, saving model to url.h5
2161/2161 [=====] - 6s 3ms/step - loss: 0.1777 - acc: 0.9467 - val_loss: 0.0850 - val_acc: 0.9766
Epoch 3/5
2151/2161 [=====>.] - ETA: 0s - loss: 0.0675 - acc: 0.9862
Epoch 3: val_acc improved from 0.97661 to 0.99124, saving model to url.h5
2161/2161 [=====] - 6s 3ms/step - loss: 0.0674 - acc: 0.9862 - val_loss: 0.0529 - val_acc: 0.9912
Epoch 4/5
2146/2161 [=====>.] - ETA: 0s - loss: 0.0508 - acc: 0.9914
Epoch 4: val_acc improved from 0.99124 to 0.99283, saving model to url.h5
2161/2161 [=====] - 6s 3ms/step - loss: 0.0507 - acc: 0.9914 - val_loss: 0.0424 - val_acc: 0.9928
Epoch 5/5
2153/2161 [=====>.] - ETA: 0s - loss: 0.0422 - acc: 0.9933
Epoch 5: val_acc did not improve from 0.99283
2161/2161 [=====] - 6s 3ms/step - loss: 0.0423 - acc: 0.9932 - val_loss: 0.0430 - val_acc: 0.9923
```

FIGURE 4.78 – Image 128

```
# la variation de Accuracy durant l'apprentissage  
plt.plot(history3.history['acc'], color='red')  
plt.plot(history3.history['val_acc'], color='green')  
plt.title('Model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train_acc','val_acc'], loc = 'upper right')  
plt.show()
```

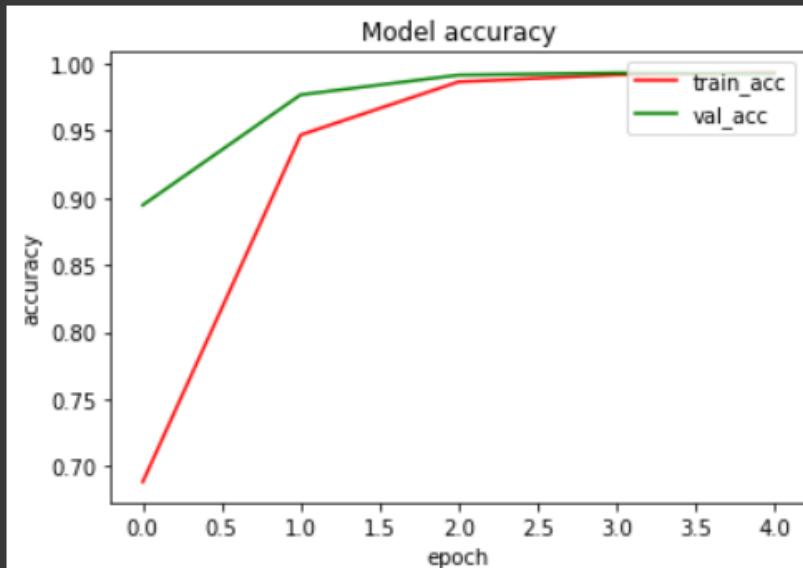


FIGURE 4.79 – Image 129

```
▶ # la variation de loss durant l'apprentissage  
plt.plot(history3.history['loss'], color ='red')  
plt.plot(history3.history['val_loss'], color ='green')  
plt.title('Model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train_loss','val_loss'], loc = 'upper right')  
plt.show()
```

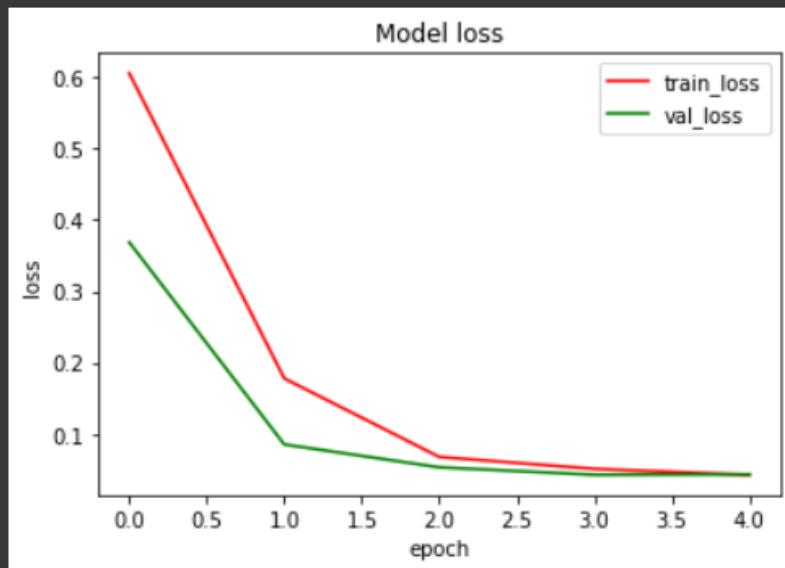


FIGURE 4.80 – Image 130

```
▶ # predicting on test data.  
pred_test3 = model3.predict(x_test)  
for i in range (len(pred_test3)):  
    if (pred_test3[i] < 0.5):  
        pred_test3[i] = 0  
    else:  
        pred_test3[i] = 1  
pred_test3 = pred_test3.astype(int)  
  
👤 4322/4322 [=====] - 6s 1ms/step  
  
[ ] view_result(pred_test3[:10])  
  
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante
```

FIGURE 4.81 – Image 131

```

▶ score3 = model3.evaluate( x_test, y_test, verbose = 0)
print(score3)

👤 [0.043009042739868164, 0.9922702312469482]

Le modèle 4
On va ajouter de couches intermédiaires

[ ] model4 = Sequential()

model4.add(Dense(32, activation = 'relu', input_shape = (16, )))
model4.add(Dense(16, activation='relu'))
model4.add(Dense(8, activation='relu'))
model4.add(Dense(64, activation='relu'))
model4.add(Dense(32, activation='sigmoid'))
model4.add(Dense(1, activation='sigmoid'))
model4.summary()

Model: "sequential_3"

Layer (type) Output Shape Param #
=====
dense_12 (Dense) (None, 32) 544
dense_13 (Dense) (None, 16) 528
dense_14 (Dense) (None, 8) 136

```

FIGURE 4.82 – Image 132

```

dense_15 (Dense) (None, 64) 576
dense_16 (Dense) (None, 32) 2080
dense_17 (Dense) (None, 1) 33
=====
Total params: 3,897
Trainable params: 3,897
Non-trainable params: 0

optimiseur4= keras.optimizers.Adam(lr=0.0001)
model4.compile(optimizer= optimiseur4 ,loss='binary_crossentropy',metrics=['acc'])

/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use super().__init__(name, **kwargs)

checkpointer4 = ModelCheckpoint('url.h5', monitor='val_acc', mode='max', verbose=2, save_best_only=True)
history4=model4.fit(x_train, y_train, batch_size=256, epochs=5, validation_data=(x_test, y_test), callbacks=[checkpointer4])

Epoch 1/5
2148/2161 [=====>.] - ETA: 0s - loss: 0.22275 - acc: 0.9100
Epoch 1: val_acc improved from -inf to 0.99466, saving model to url.h5
2161/2161 [=====] - 8s 4ms/step - loss: 0.2263 - acc: 0.9105 - val_loss: 0.0303 - val_acc: 0.9947
Epoch 2/5
2159/2161 [=====>.] - ETA: 0s - loss: 0.0239 - acc: 0.9952

```

FIGURE 4.83 – Image 133

```

Epoch 1: val_acc improved from -inf to 0.99466, saving model to url.h5
2161/2161 [=====>.] - 8s 4ms/step - loss: 0.2263 - acc: 0.9105 - val_loss: 0.0303 - val_acc: 0.9947
Epoch 2/5
2159/2161 [=====>.] - ETA: 0s - loss: 0.0239 - acc: 0.9952
Epoch 2: val_acc improved from 0.99466 to 0.99638, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0239 - acc: 0.9952 - val_loss: 0.0180 - val_acc: 0.9964
Epoch 3/5
2148/2161 [=====>.] - ETA: 0s - loss: 0.0186 - acc: 0.9959
Epoch 3: val_acc improved from 0.99638 to 0.99664, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0186 - acc: 0.9960 - val_loss: 0.0160 - val_acc: 0.9966
Epoch 4/5
2149/2161 [=====>.] - ETA: 0s - loss: 0.0171 - acc: 0.9963
Epoch 4: val_acc did not improve from 0.99664
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0171 - acc: 0.9963 - val_loss: 0.0166 - val_acc: 0.9965
Epoch 5/5
2149/2161 [=====>.] - ETA: 0s - loss: 0.0165 - acc: 0.9964
Epoch 5: val_acc improved from 0.99664 to 0.99681, saving model to url.h5
2161/2161 [=====>.] - 7s 3ms/step - loss: 0.0164 - acc: 0.9964 - val_loss: 0.0149 - val_acc: 0.9968

[ ] # la variation de Accuracy durant l'apprentissage
plt.plot(history4.history['acc'], color='red')
plt.plot(history4.history['val_acc'], color='green')
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train_acc', 'val_acc'], loc = 'upper right')
plt.show()

```

FIGURE 4.84 – Image 134

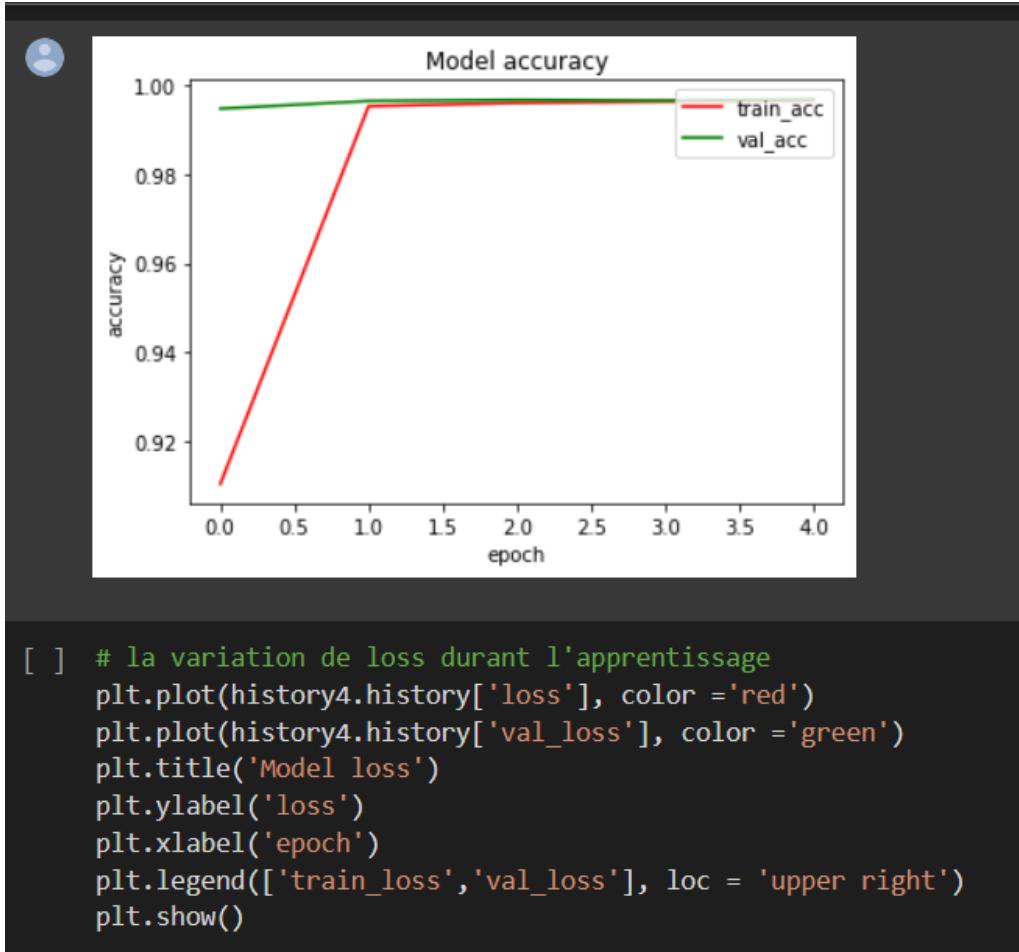


FIGURE 4.85 – Image 135

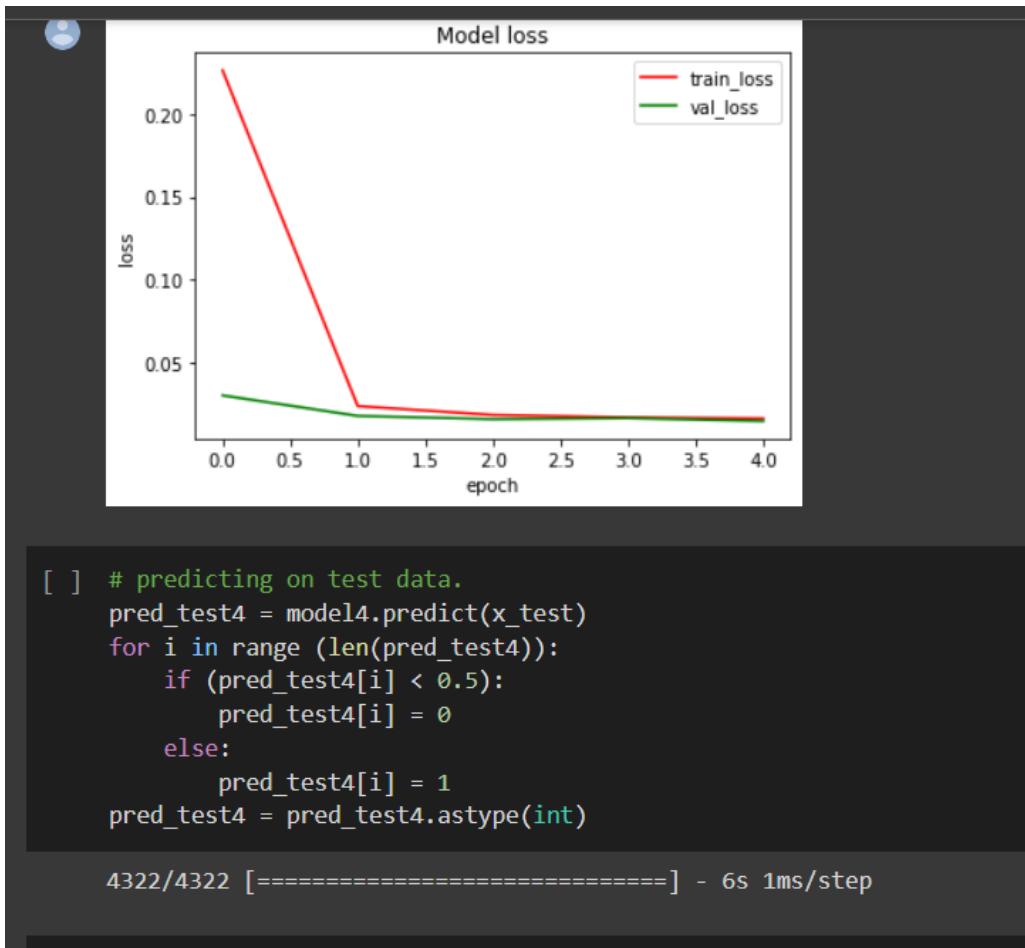


FIGURE 4.86 – Image 136

```
[ ] view_result(pred_test4[:10])
```

```
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante
```

```
[ ] view_result(y_test[:10])
```

```
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Non malveillante  
Non malveillante  
Non malveillante  
Malveillante  
Malveillante  
Malveillante
```

FIGURE 4.87 – Image 137

```
▶ score4 = model4.evaluate( x_test, y_test, verbose = 0)  
print(score4)  
👤 [0.01491992361843586, 0.9968112111091614]
```

FIGURE 4.88 – Image 138

Bibliographie

- [1] Kaggle. <<https://www.kaggle.com/datasets>>.
- [2] Scikit. <<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>>.
- [3] Scikit. <https://scikit-learn.org/stable/modules/naive_bayes.html>.
- [4] Scikit. <<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>>.
- [5] Scikit. <<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>>.