CYCLE D'INGENIEURE EN GENIE INFORMATIQUE

Année universitaire : 2025-2026

Business Intelligence

Rapport Mini Projet : Détection de sites de phishing avec XGBoost

Réalisé par :

Wijdane Oubhat
Salma Amalik

Sous l'encadrement de :

Objectif du projet

L'objectif principal est de classer les sites web en deux catégories : légitime (safe) et phishing (malveillant). Pour cela, nous utiliserons les caractéristiques du site (features) comme la structure de l'URL, le contenu de la page, ou la présence du protocole HTTPS afin d'entraîner un modèle capable d'identifier les sites suspects avec une forte précision.

Algorithme utilisé : XGBoost

XGBoost (Extreme Gradient Boosting) est un algorithme de Machine Learning supervisé basé sur les arbres de décision. Son principe repose sur la création d'une série d'arbres, où chaque nouvel arbre vient corriger les erreurs du précédent. Grâce à cette approche, le modèle devient de plus en plus précis à chaque itération.

Avantages de XGBoost :

- Haute performance et rapidité
- Réduction du surapprentissage (overfitting)
- Gestion efficace des grandes bases de données

Résultats attendus

À l'issue du projet, nous obtiendrons un modèle fiable et précis capable d'identifier automatiquement les sites de phishing, de réduire les risques liés aux cyberattaques et de contribuer à la sécurité des utilisateurs sur Internet.

Dataset:

Descriptions de toutes les caractéristiques du dataset

1. Caractéristiques liées à l'URL

- **FILENAME** : Nom du fichier associé à l'URL, souvent utilisé comme identifiant technique.
- **URL** : L'adresse web complète.
- **URLLength** : Longueur totale de l'URL ; une URL très longue est souvent suspecte.
- **Domain** : Nom de domaine extrait de l'URL.
- **DomainLength** : Longueur totale du domaine ; les domaines trop longs sont souvent générés automatiquement.
- **IsDomainIP** : Indique si le domaine est une adresse IP (1) ou un nom de domaine normal (0). Les phishing utilisent souvent des IP pour masquer leur identité.
- **TLD** : Extension du domaine (ex : .com, .net, .xyz).
- **URLSimilarityIndex** : Mesure la similarité entre l'URL et une URL légitime connue, pour détecter les imitations.
- **CharContinuationRate** : Mesure la présence de répétitions de caractères (comme "aaa", "////"), signe d'obfuscation.
- **TLDLegitimateProb** : Probabilité que le TLD corresponde à un domaine fiable.
- **URLCharProb** : Score indiquant si les caractères de l'URL ressemblent à ceux que l'on retrouve dans une URL normale.
- **TLDLength** : Longueur de l'extension du domaine.
- **NoOfSubDomain** : Nombre de sous-domaines présents dans l'URL.
- **HasObfuscation** : Indique si l'URL contient des éléments d'obfuscation (encodage, caractères masqués…).
- **NoOfObfuscatedChar** : Nombre total de caractères obfusqués dans l'URL.
- **ObfuscationRatio** : Proportion des caractères obfusqués par rapport à la longueur totale de l'URL.
- **NoOfLettersInURL** : Nombre de lettres dans l'URL.
- **LetterRatioInURL** : Pourcentage de lettres dans l'URL.
- **NoOfDegitsInURL** : Nombre total de chiffres dans l'URL.
- **DegitRatioInURL** : Pourcentage de chiffres dans l'URL ; les URLs frauduleuses contiennent souvent beaucoup de chiffres.
- **NoOfEqualsInURL** : Nombre de symboles « = ».
- **NoOfQMarkInURL** : Nombre de symboles « ? ».
- **NoOfAmpersandInURL** : Nombre de symboles « & ».

- **NoOfOtherSpecialCharsInURL** : Nombre d'autres caractères spéciaux dans l'URL.
- **SpacialCharRatioInURL** : Proportion totale de caractères spéciaux.
- **IsHTTPS** : Indique si l'URL utilise le protocole HTTPS ou non.

2. Caractéristiques liées au code source HTML de la page

- **LineOfCode** : Nombre total de lignes dans le code source ; les pages phishing sont souvent très simples et limitées.
- **LargestLineLength** : Longueur de la plus grande ligne du code source.
- **HasTitle** : Indique la présence d'une balise <title>.
- **Title** : Contenu textuel du titre de la page.
- **DomainTitleMatchScore** : Score de similarité entre le domaine et le titre.
- **URLTitleMatchScore** : Similarité entre l'URL et le titre de la page.
- **HasFavicon** : Indique si un favicon est présent.
- **Robots** : Présence d'un fichier robots.txt.
- **IsResponsive** : Indique si la page est responsive ; les pages légitimes le sont généralement.
- **NoOfURLRedirect** : Nombre total de redirections effectuées par la page.
- **NoOfSelfRedirect** : Redirections vers la même URL.
- **HasDescription** : Présence d'une meta description.
- **NoOfPopup** : Nombre de pop-ups détectés sur la page.
- **NoOfiFrame** : Nombre de balises <iframe>, souvent utilisées en phishing pour masquer du contenu.
- **HasExternalFormSubmit** : Indique si un formulaire envoie les données vers un domaine externe (typique du vol d'identifiants).
- **HasSocialNet** : Présence de liens vers des réseaux sociaux.
- **HasSubmitButton** : Présence d'un bouton de soumission dans un formulaire.
- **HasHiddenFields** : Présence de champs cachés dans la page.
- **HasPasswordField** : Présence d'un champ « password » dans un formulaire.

3. Mots clés présents dans la page

- **Bank** : Présence de mots liés aux banques.
- **Pay** : Présence de mots liés aux paiements.
- **Crypto** : Termes liés aux cryptomonnaies.

- **HasCopyrightInfo** : Indique si la page contient une mention de copyright.

4. Statistiques d'éléments HTML

- **NoOfImage** : Nombre total d'images dans la page.
- **NoOfCSS** : Nombre de fichiers CSS utilisés.
- **NoOfJS** : Nombre de fichiers JavaScript.
- **NoOfSelfRef** : Nombre de liens internes vers le même domaine.
- **NoOfEmptyRef** : Liens vides (du type « # »).
- **NoOfExternalRef** : Liens vers des domaines externes.

5. Label (variable cible)

- **label** :
  - **0** = site légitime
  - **1** = site phishing

PRÉTRAITEMENT

3 - PRÉTRAITEMENT DES VARIABLES CATÉGORIELLES

L'objectif de cette phase est double :

1. Éliminer les colonnes non informatives qui pourraient introduire du bruit ou des biais dans le modèle
2. Transformer les données catégorielles en format numérique tout en préservant leur signification sémantique

Le traitement se concentre particulièrement sur les attributs liés aux URLs et domaines, où les informations textuelles brutes sont remplacées par leurs métriques dérivées déjà calculées, plus adaptées à l'apprentissage automatique.

```
from google.colab import files
uploaded = files.upload()
```

Sélect. fichiers PhiUSIIL_...Dataset.csv

PhiUSIIL_Phishing_URL_Dataset.csv(text/csv) - 56854345 bytes, last modified: 01/12/2025 - 100% done

Saving PhiUSIIL_Phishing_URL_Dataset.csv to PhiUSIIL_Phishing_URL_Dataset.csv

```
import pandas as pd

df = pd.read_csv(list(uploaded.keys())[0])
df.head()
```

	FILENAME	URL	URLLength	Domain	DomainLength	IsDomainIP	TLD	URLSimilarityIndex	CharContir
0	521848.txt	https://www.southbankmosaics.com	31	www.southbankmosaics.com	24	0	com	100.0	
1	31372.txt	https://www.uni-mainz.de	23	www.uni-mainz.de	16	0	de	100.0	
2	597387.txt	https://www.voicefmradio.co.uk	29	www.voicefmradio.co.uk	22	0	uk	100.0	
3	554095.txt	https://www.sfnmjournal.com	26	www.sfnmjournal.com	19	0	com	100.0	

=> on affiche que les colonnes catecorielles :

```
def divide_by_type_columns(df,tyype):
    df_copy = df.select_dtypes(include = [f'{tyype}'])
    return df_copy
```

```
char_columns = divide_by_type_columns(df, 'object')
```

```
char_columns.head()
```

	FILENAME	URL	Domain	TLD	Title
0	521848.txt	https://www.southbankmosaics.com	www.southbankmosaics.com	com	à„à¹ˆà¸²à¸§à¸²à¸" à„à¹ˆà¸²à¸§à¸§à¸±à¸™à¸™...
1	31372.txt	https://www.uni-mainz.de	www.uni-mainz.de	de	johannes gutenberg-universitÃ¤t mainz
2	597387.txt	https://www.voicefmradio.co.uk	www.voicefmradio.co.uk	uk	voice fm southampton
3	554095.txt	https://www.sfnmjournal.com	www.sfnmjournal.com	com	home page: seminars in fetal and neonatal medi...
4	151578.txt	https://www.rewildingargentina.org	www.rewildingargentina.org	org	fundaciÃ³n rewilding argentina

1. FILENAME

- Le nom du fichier ne donne aucune information sur le site.
- => Supprimer la colonne

df = df.drop(columns=['FILENAME'])

2. URL

- Pour un modèle ML , on ne garde pas l'URL brute, car les features numériques dérivées existent déjà :
  URLLength, URLCharProb, URLSimilarityIndex, etc.
- => on supprime la colonne pour éviter les problèmes :

df = df.drop(columns=['URL'])

3.Domain

- Même principe : les informations utiles sont déjà dans DomainLength, IsDomainIP, TLD, etc.
- Donc on peut aussi supprimer :

df = df.drop(columns=['Domain'])

4.Title

- Déjà transformée dans les colonnes :

  HasTitle, URLTitleMatchScore, DomainTitleMatchScore
- Si tu veux utiliser NLP pour extraire plus d'infos, tu peux garder Title.
- Sinon, tu peux supprimer :

df = df.drop(columns=['Title'])

5. TLD

La colonne TLD (Top-Level Domain) est catégorielle, par exemple : com, uk, org, net, etc.

- Certains TLD sont plus utilisés pour les sites légitimes (.com, .org)
- D'autres peuvent être plus suspects (.xyz, .top)

on encode la colonne pour que le modèle ML puisse la comprendre.

=> Label Encoding (pour RandomForest, XGBoost…)

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df['TLD'] = encoder.fit_transform(df['TLD'])

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
df['TLD'] = encoder.fit_transform(df['TLD'])
```

```
df.head()
```

	URLLength	DomainLength	IsDomainIP	TLD	URLSimilarityIndex	CharContinuationRate	TLDLegitimateProb	URLCharProb	TLDLength	NoOfSubDomai
0	31	24	0	231	100.0	1.000000	0.522907	0.061933	3	
1	23	16	0	254	100.0	0.666667	0.032650	0.050207	2	
2	29	22	0	647	100.0	0.866667	0.028555	0.064129	2	
3	26	19	0	231	100.0	1.000000	0.522907	0.057606	3	
4	33	26	0	503	100.0	1.000000	0.079963	0.059441	3	

5 rows × 52 columns

Mise en place du modèle XGBoost, entraînement sur les données et génération des premiers résultats :

Dans cette section, nous présentons le processus de mise en place et d'entraînement d'un modèle XGBoost pour la classification. Nous commençons par installer et importer les bibliothèques nécessaires, puis nous téléversons et explorons le fichier de données Excel afin de comprendre sa structure et ses dimensions. Ensuite, nous préparons les données en séparant les features de la variable cible, et nous réalisons une division du dataset en ensembles d'entraînement et de test tout en conservant la distribution des classes. Après cette préparation, nous construisons le modèle XGBoost avec des paramètres initiaux et l'entraînons sur les données d'entraînement. Enfin, nous effectuons des prédictions sur l'ensemble de test et présentons un aperçu des résultats en comparant les premières prédictions avec les valeurs réelles, ce qui nous permet d'évaluer visuellement les performances initiales du modèle.

1 - IMPORTATION DES BIBLIOTHÈQUES

Cette section importe toutes les bibliothèques nécessaires XGBoost pour le modèle, scikit-learn pour le préprocessing et pandas/numpy pour la manipulation des données

```
!pip install xgboost pandas numpy scikit-learn matplotlib openpyxl -q

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import xgboost as xgb
import matplotlib.pyplot as plt
```

2 - CHARGEMENT DU DATASET

Téléversement et lecture du fichier Excel depuis le PC

Conversion en DataFrame pandas pour analyse

Vérification de la structure et dimensions

```
# 2. TÉLÉVERSEMENT DU FICHIER EXCEL
from google.colab import files
uploaded = files.upload()
```

```
Sélectionnez votre fichier Excel (.xlsx) depuis votre PC...
Sélect. fichiers  Aucun fichier choisi  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving fichier_concatene.xlsx to fichier_concatene.xlsx
```

```
import io

filename = list(uploaded.keys())[0]
df = pd.read_excel(io.BytesIO(uploaded[filename]))


print(f"📊 Dimensions : {df.shape[0]} lignes, {df.shape[1]} colonnes")

print(df.head())
```

```
📊 Dimensions : 235795 lignes, 52 colonnes
   URLLength  DomainLength  IsDomainIP  TLDLength  NoOfSubDomain  \
0   1.496434            24           0          3              1
1   1.429846            16           0          2              1
2   1.481877            22           0          2              2
3   1.457646            19           0          3              1
4   1.509918            26           0          3              1

   HasObfuscation  NoOfObfuscatedChar  NoOfLettersInURL  NoOfDegitsInURL  \
0               0                 0.0          1.372307              0.0
1               0                 0.0          1.194706              0.0
2               0                 0.0          1.327761              0.0
3               0                 0.0          1.291725              0.0
4               0                 0.0          1.397363              0.0

   NoOfEqualsInURL  ...  URLSimilarityIndex  CharContinuationRate  \
0                0  ...               100.0              1.000000
1                0  ...               100.0              0.666667
2                0  ...               100.0              0.866667
3                0  ...               100.0              1.000000
4                0  ...               100.0              1.000000
```

```
   NoOfEqualsInURL  ...  URLSimilarityIndex  CharContinuationRate  \
0                0  ...               100.0              1.000000
1                0  ...               100.0              0.666667
2                0  ...               100.0              0.866667
3                0  ...               100.0              1.000000
4                0  ...               100.0              1.000000

   TLDLegitimateProb  URLCharProb  ObfuscationRatio  LetterRatioInURL  \
0           0.522907     0.061933                 0             0.581
1           0.032650     0.050207                 0             0.391
2           0.028555     0.064129                 0             0.517
3           0.522907     0.057606                 0             0.500
4           0.079963     0.059441                 0             0.606

   DegitRatioInURL  SpacialCharRatioInURL  DomainTitleMatchScore  \
0                0                  0.032               0.000000
1                0                  0.087              55.555556
2                0                  0.069              46.666667
3                0                  0.038               0.000000
4                0                  0.030             100.000000

   URLTitleMatchScore
0            0.000000
1           55.555556
2           46.666667
3            0.000000
4          100.000000

[5 rows x 52 columns]
```

PRÉPARATION DES DONNÉE :

Séparation des features (X) et de la target (y)

La variable 'label' est notre cible de classification

51 features sont utilisées pour prédire 2 classes

```
target_column = 'label'

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

print(f"Features: {X.shape}")
print(f"Target: {y.shape}")
```

```
Features: (235795, 51)
Target: (235795,)
```

DIVISION ENTRAÎNEMENT/TEST

Séparation en 80% pour l'entraînement, 20% pour le test

Stratification pour conserver les proportions de classes

Random state fixé pour reproductibilité des résultats

```
om sklearn.model_selection import train_test_split

train, X_test, y_train, y_test = train_test_split(
  X, y,
  test_size=0.2,        # 20% pour le test
  random_state=42,      # Pour la reproductibilité
  stratify=y            # Même distribution des classes


int(f"✅ Division terminée")
int(f"📊 Train: {X_train.shape[0]} échantillons")
int(f"🧪 Test: {X_test.shape[0]} échantillons")
```

```
Division terminée
Train: 188636 échantillons
Test: 47159 échantillons
```

CONFIGURATION DU MODÈLE XGBOOST

Définition des hyperparamètres du modèle

1. n_estimators=100

C'est une valeur standard de départ. Assez d'arbres pour capter les patterns, mais pas trop pour éviter de sur-apprendre ou d'être trop lent. Un bon compromis entre performance et temps de calcul.

2. max_depth=3

Pour garder le modèle simple et interprétable. Une profondeur limitée évite le sur-apprentissage et produit des règles de décision compréhensibles..

3. learning_rate=0.1

C'est la valeur par défaut recommandée.

4. random_state=42

Pour la reproductibilité. C’est une convention. N'importe quel nombre fixe ferait l'affaire. Cela garantit que les résultats sont les mêmes à chaque exécution.

5. use_label_encoder=False

C'est obligatoire dans les nouvelles versions de XGBoost. Cela évite un warning. L'encodage des labels est maintenant géré automatiquement.

6. eval_metric='logloss'

C'est la métrique standard pour la classification binaire. Elle mesure l'erreur de prédiction probabiliste et est plus informative que l'accuracy simple.

```
import xgboost as xgb
model = xgb.XGBClassifier(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)


print("Entraînement...")
model.fit(X_train, y_train)
```

```
Entraînement...
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [19:23:12] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

ENTRAÎNEMENT XGBOOST

entraînement sur les données d'apprentissage

```
import xgboost as xgb
model = xgb.XGBClassifier(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)


print("Entraînement...")
model.fit(X_train, y_train)
```

```
Entraînement...
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [19:23:12] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

PRÉDICTIONS SUR L'ENSEMBLE DE TEST

Application du modèle XGBoost entraîné aux données de test pour obtenir les prédictions

```
y_pred = model.predict(X_test)
print("Prédictions terminées")
print(f"Exemple : {y_pred[:5]}")
```

```
Prédictions terminées
mple : [1 1 1 1 1]
```

COMPARAISON DES 5 PREMIÈRES PRÉDICTIONS

Aperçu rapide de la performance du modèle sur les premiers échantillons de test

```
for i in range(5):
    vrai = y_test.iloc[i] if hasattr(y_test, 'iloc') else y_test[i]
    pred = y_pred[i]
    status = "✓ CORRECT" if vrai == pred else "✗ ERREUR"
    vrai_label = "Légitime" if vrai == 0 else "Suspect"
    pred_label = "Légitime" if pred == 0 else "Suspect"

    print(f"{i+1:<3} {vrai_label:<6} {pred_label:<6} {status:<10}")

print("-" * 45)
```

```
🔍 Comparaison des 5 premiers échantillons :
---------------------------------------------
#   VRAI   PRÉDIT STATUT
---------------------------------------------
1   Suspect Suspect ✓ CORRECT
2   Suspect Suspect ✓ CORRECT
3   Suspect Suspect ✓ CORRECT
4   Suspect Suspect ✓ CORRECT
5   Suspect Suspect ✓ CORRECT
---------------------------------------------
```