UNIVERSITE MOLAY ISMAIL
FACULTE DES SCIENCES ET TECHNIQUE
ERRACHIDIA

MODULE : INTELLIGENCE ARTIFICIELLE

2eme année cycle d'ingénieur

Génie informatique & Business intelligence

Analyse et classification des tweets pour identifier le cyberbullying

---

Réalisé par : Aalali Ayoub

Encadré par : Pr. Imad Zeroual

ANNEE UNIVERSITAIRE : 2024/2025

REMERCIEMENTS

Au terme de ce mini-projet du module d'Intelligence Artificielle, intitulé « Analyse et classification des tweets pour identifier le cyberbullying », je tiens avant tout à remercier Dieu, qui m'a accordé la force, la patience et la persévérance nécessaires pour mener à bien ce travail.

J'exprime ma profonde reconnaissance à mes parents pour leur amour inconditionnel, leur soutien constant et leur confiance tout au long de mon parcours académique. Leur présence et leurs encouragements ont été essentiels pour franchir cette étape.

Je remercie sincèrement Pr Imad Zeroual, mon encadrant, pour sa disponibilité, ses conseils précieux et son accompagnement tout au long de ce projet. Son expertise et sa rigueur ont grandement contribué à la qualité de ce travail.

Je n'oublie pas de remercier chaleureusement mes camarades et amis pour leur soutien moral, leurs partages d'idées et leur bienveillance. Un grand merci également à mes enseignants, dont les cours ont constitué une base solide pour la réalisation de ce projet.

Enfin, je remercie toutes les personnes qui, de près ou de loin, m'ont encouragé, conseillé ou soutenu dans cette démarche. Leur aide m'a été précieuse pour concrétiser ce travail et enrichir mon expérience dans le domaine de l'intelligence artificiell

Table des matières

1. Introduction

Dans le contexte actuel des réseaux sociaux, la multiplication des échanges en ligne a entraîné une recrudescence des contenus à caractère offensant ou discriminatoire. Les utilisateurs publient quotidiennement des messages, tweets et commentaires dont certains véhiculent du cyberharcèlement (cyberbullying) : attaques basées sur l'âge, le genre, l'ethnie ou la religion, ainsi que des propos insultants et dénigrants non ciblés. Lorsqu'ils sont diffusés sans filtre, ces contenus nuisent au bien-être des individus, détériorent le climat général de la plateforme et exposent celle-ci à des risques d'atteinte à sa réputation et à d'éventuelles conséquences juridiques.

Le présent rapport s'attache à proposer une solution automatique de détection et de classification des publications textuelles contenant du cyberharcèlement. L'objectif est de créer et d'entraîner un modèle de Deep Learning capable d'identifier, au sein de chaque nouveau tweet ou commentaire, la présence d'un message offensant correspondant à l'une des catégories suivantes :

- ethnicity : attaque visant l'origine ethnique ou raciale,
- religion : propos moqueurs ou hostiles à l'égard d'une croyance ou d'une communauté religieuse,
- age : remarque discriminatoire fondée sur l'âge,
- gender : insulte ou stéréotype lié au genre,
- other_cyberbullying : harcèlement général non rattaché aux catégories précédentes,
- not_cyberbullying : absence de contenu harcelant.

Pour ce faire, le projet repose sur un jeu de données annoté provenant de la plateforme Kaggle, comportant pour chaque instance : le texte du tweet (ou du commentaire) et l'étiquette de cyberharcèlement correspondante. Après un processus rigoureux de prétraitement des données (nettoyage des textes, suppression des éléments non pertinents, tokenisation, vectorisation et padding), les textes sont utilisés pour entraîner un réseau de type séquentiel : couche d'Embedding, couche récurrente (LSTM ou GRU), Dropout, puis couche de sortie dense à activation softmax. Le modèle est compilé avec la fonction de perte sparse_categorical_crossentropy et l'optimiseur Adam, puis évalué sur un jeu de test stratifié afin de mesurer sa capacité de généralisation (accuracy, précision, rappel, F1-score et matrice de confusion).

Enfin, pour rendre la solution opérationnelle, le modèle entraîné, le tokenizer et l'encodeur de labels sont sauvegardés. Une API (déployée en environnement Colab puis exposée via ngrok) permet de recevoir en requête POST un texte à évaluer et de retourner la prédiction de classe. En phase finale, cette API peut être intégrée à une plateforme sociale (par exemple via un front-end Next.js) pour bloquer automatiquement toute publication cataloguée comme cyberharcèlement. Ainsi, l'ensemble du processus garantit qu'un message jugé offensant ne soit pas diffusé, contribuant à créer un espace en ligne plus sain et sécurisé pour tous les utilisateurs.

2. Description et enjeux des données

2-1. Source des données

Les données utilisées dans ce projet proviennent de la plateforme Kaggle, qui héberge des jeux de données publics pour l'analyse de données et le machine learning. Le dataset sélectionné est intitulé "Cyberbullying Classification", et il est conçu pour entraîner des modèles capables de détecter différents types de cyberintimidation sur les réseaux sociaux, en particulier sur Twitter.

2-2. Structure des données

Le dataset utilisé se compose de **colonnes principales** représentant le contenu des tweets et leurs catégories associées. Sa structure est décrite comme suit :

Colonne	Type de données	Description
Tweet_text	Texte (string)	Contenu textuel du tweet tel qu'il a été publié sur Twitter.
Cyberbullying_type	Catégorie (string)	Étiquette indiquant le type de cyberintimidation associé au tweet.

La colonne cyberbullying_type peut prendre l'une des valeurs suivantes :

- age : cyberintimidation basée sur l'âge.
- ethnicity : propos discriminatoires liés à l'origine ethnique.
- gender : attaques ou moqueries basées sur le genre.
- religion : propos offensants en lien avec des croyances religieuses.
- other_cyberbullying : autres types d'intimidation non classés.
- not_cyberbullying : contenu jugé non offensant.

3. Pré-traitement des données

3-1. Nettoyage de texte

Lors de cette phase, j'ai appliqué plusieurs fonctions et expressions régulières afin de nettoyer les données textuelles.

Les principales étapes suivies durant ce pré-traitement sont les suivantes :

3-1-1. Traitement des valeurs manquantes

Pour détecter les valeurs nulles ou manquantes, j'ai utilisé la fonction `isnull()` combinée à `sum()` afin de calculer le nombre total de valeurs nulles dans chaque colonne. Heureusement, le jeu de données ne contient aucune valeur manquante.

```
dataset.isnull().sum()
```

	0
tweet_text	0
cyberbullying_type	0

3-1-2 : Détection et suppression des doublons

La détection des doublons a été effectuée à l'aide de la fonction **duplicated()** et la suppression de ces lignes par la fonction **drop_duplicates()**

```
dataset["tweet_text"].duplicated().sum()
dataset.drop_duplicates(subset="tweet_text", keep="first", inplace=True)
```

3-1-3. Conversion en minuscules

Cette étape permet d'uniformiser le texte en considérant les mots comme "Cyber" et "cyber" comme identiques, ce qui évite la création de doublons. Elle contribue également à réduire la dimensionnalité en limitant le nombre de mots distincts à analyser par le modèle.

Pour cela, j'ai utilisé la fonction `lower()` afin de convertir l'ensemble du texte en minuscules.

3-1-4. Suppression des URLs dans les tweets

Les URLs présentes dans les tweets n'apportent généralement pas d'informations utiles pour l'analyse du contenu textuel. Leur suppression permet de réduire le bruit dans les données, d'améliorer la qualité du texte traité, et donc d'optimiser la performance du modèle de classification.

```python
import re
def supprimer_urls(texte):
    return re.sub(r"http\S+|www\S+|https\S+", "", str(texte), flags=re.MULTILINE)
dataset["tweet_text"] = dataset["tweet_text"].apply(supprimer_urls)
```

3-1-5. Suppression des mentions

Les mentions correspondent à des noms d'utilisateurs spécifiques qui n'apportent pas de valeur sémantique pour la classification du contenu. Les conserver peut introduire du bruit et biaiser le modèle, surtout si certains utilisateurs sont surreprésentés. Supprimer les mentions permet donc de se concentrer sur le contenu réel du message, améliorant ainsi la qualité des données et la précision du modèle.

```python
def supprimer_mentions(texte):
    return re.sub(r"@\w+", "", str(texte))
dataset["tweet_text"] = dataset["tweet_text"].apply(supprimer_mentions)
```

3-1-6. Suppression des valeurs numériques

Les valeurs numériques (comme des dates, quantités ou codes) n'apportent souvent pas d'information pertinente pour la détection de cyberharcèlement. Elles peuvent créer du bruit dans les données et augmenter inutilement la complexité du vocabulaire. En les supprimant, on simplifie le texte et on permet au modèle de mieux se concentrer sur les éléments linguistiques porteurs de sens.

```python
def supprimer_nombres(texte):
    return re.sub(r"\d+", "", str(texte))

dataset["tweet_text"] = dataset["tweet_text"].apply(supprimer_nombres)
```

3-1-7. Suppression des ponctuations et des caractère spéciaux

Les ponctuations et caractères spéciaux n'ont généralement pas de valeur sémantique utile pour l'analyse du contenu textuel dans le contexte de la classification de cyberharcèlement. Leur présence peut perturber la tokenisation et introduire du bruit dans les données. Les supprimer permet de nettoyer les textes, d'harmoniser les données, et d'améliorer la qualité de l'apprentissage du modèle.

```python
def supprimer_ponctuation(texte):
    return re.sub(r"[^\w\s]", "", str(texte))

dataset["tweet_text"] = dataset["tweet_text"].apply(supprimer_ponctuation)


dataset.loc[:, "tweet_text"] = dataset["tweet_text"].str.replace(
    r"[#@!%?\*&$€+=\[\]\(\)\{\}<>\\/:;\"'\|]", "", regex=True
)
```

3-1-8. Suppression des mots vides (Stop Words)

Les *stop words* en anglais (tels que *"the"*, *"is"*, *"at"*, *"on"*, *"a"*, *"and"*) sont des mots très fréquents qui n'apportent pas d'information utile pour différencier les types de cyberharcèlement. Leur présence peut augmenter inutilement la taille du vocabulaire et introduire du bruit dans les données. En les supprimant, on améliore la qualité des représentations textuelles, ce qui aide le modèle à mieux se concentrer sur les mots significatifs liés au contenu offensant ou discriminatoire.

```python
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')

stop_words = set(stopwords.words('english'))

def remove_stopwords(text):
    words = str(text).split()
    filtered = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered)

dataset.loc[:, "tweet_text"] = dataset["tweet_text"].apply(remove_stopwords)
```

3-1-9. Application du Stemming

Le *stemming* consiste à réduire les mots à leur racine ou forme de base (par exemple, "playing", "played", "plays" deviennent "play"). Cette étape permet :

- De **réduire la dimensionnalité du vocabulaire**,
- De **regrouper les variantes d'un même mot** sous une seule forme,
- Et donc **d'améliorer la cohérence des représentations textuelles**.

Cela aide le modèle à généraliser davantage, car des mots ayant le même sens seront traités comme équivalents.

```python
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')

ps = PorterStemmer()

def stem_text(text):
    words = str(text).split()
    stemmed_words = [ps.stem(word) for word in words]
    return ' '.join(stemmed_words)

dataset.loc[:, "tweet_text"] = dataset["tweet_text"].apply(stem_text)
```

3-2. Tokenisation

La tokenisation est un processus en traitement automatique du langage naturel (TALN) qui consiste à diviser un texte en unités de base appelées *tokens*. Ces unités peuvent être des mots, des phrases, ou des caractères, selon le niveau d'analyse souhaité.

C'est une étape fondamentale qui prépare le texte pour l'analyse, l'extraction d'information ou l'entraînement d'un modèle de machine Learning ou Deep Learning.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")
tokenizer.fit_on_texts(dataset["tweet_text"])
```

Paramètres :

- num_words=10000 :

Ce paramètre signifie que **le tokenizer ne gardera que les 10 000 mots les plus fréquents** dans le corpus.

➤ Cela permet de réduire la taille du vocabulaire, ce qui améliore l'efficacité et évite que le modèle ne soit perturbé par des mots très rares.

- oov_token="<OOV>" :

Ce paramètre définit un token spécial (Out-Of-Vocabulary) qui sera utilisé à la place des mots inconnus (non vus pendant l'entraînement).

➤ Cela évite les erreurs si un mot nouveau apparaît dans les données de test ou d'utilisation.

3-3. Vectorisation

La vectorisation est le processus qui consiste à transformer un texte (ou une séquence de mots) en vecteurs numériques compréhensibles par un modèle de machine learning ou deep learning.

```
sequences = tokenizer.texts_to_sequences(dataset["tweet_text"])
```

Exemple :

```
print(sequences[12])
```

```
[27, 1327, 2370, 4, 710, 993]
```

3-4. Padding

Le padding est une technique qui consiste à compléter les séquences de tokens (entiers) avec des valeurs neutres (souvent 0) afin de leur donner toute la même longueur.

Pourquoi padding ?

Les réseaux de neurones (comme les LSTM ou CNN) ne peuvent pas traiter des séquences de longueurs différentes dans un même batch.

- Le padding permet donc de :
- Uniformiser la taille des séquences.
- Faciliter l'entraînement du modèle.
- Préserver la structure du texte en respectant l'ordre des mots.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

padded_sequences = pad_sequences(sequences, maxlen=50, padding='post', truncating='post')
```

Paramètres :

sequences :

- C'est la liste des séquences de tokens (entiers) générées par tokenizer.texts_to_sequences(...).

maxlen=50 :

- Fixe la **longueur maximale** de chaque séquence à **50**.
- Si une séquence contient **plus de 50 tokens**, elle sera **coupée**.
- Si elle en contient **moins**, elle sera **complétée** avec des 0.

padding='post' :

- Les **zéros** seront ajoutés **à la fin** (post) des séquences courtes.
- (alternatif : padding='pre' → les zéros sont ajoutés au début).

truncating='post' :

- Les séquences trop longues seront **coupées à la fin**.
- (alternatif : truncating='pre' → on coupe au début).

3-5. Préparation des étiquettes

Lors de cette étape, j'ai utilisé Label Encoder pour convertir les catégories de la colonne cyberbullying en valeurs entières.

```
[ ]  from sklearn.preprocessing import LabelEncoder

     label_encoder = LabelEncoder()
     dataset["label"] = label_encoder.fit_transform(dataset["cyberbullying_type"])
```

Afficher la sortie masquée

```
     for i, label in enumerate(label_encoder.classes_):
         print(f"{i} → {label}")
```

```
0 → age
1 → ethnicity
2 → gender
3 → not_cyberbullying
4 → other_cyberbullying
5 → religion
```

3-6. Séparation du jeu de données

Dans ce code, j'ai séparé mon jeu de données en deux parties : 80 % pour l'entraînement du modèle et 20 % pour le test. J'utilise la fonction train_test_split de scikit-learn pour diviser les séquences de tweets (X) et leurs labels encodés (y). J'ai fixé un random_state pour garantir la reproductibilité et j'ai stratifié la division pour que la répartition des classes soit équilibrée dans les deux ensembles.

```
from sklearn.model_selection import train_test_split

# X = les séquences de tweets, y = les labels encodés
X = padded_sequences
y = dataset["label"]

# Séparation 80% entraînement / 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

4. Construction de modèle

Dans cette étape, j'ai conçu une architecture de réseau de neurones adaptée à la classification de texte. Le but est d'apprendre à partir des séquences de mots vectorisées pour prédire la catégorie de cyberintimidation à laquelle appartient un tweet.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, GRU, SimpleRNN, Dense, Dropout
vocab_size = 10000
embedding_dim = 64
max_length = 50
rnn_units = 64
dropout_rate = 0.3
output_classes = 6
# Construction du modèle
model = Sequential()
# 1. Couche d'embedding
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim))

# 2. Couche récurrente (choisir une seule)
model.add(LSTM(rnn_units, return_sequences=False))

# 3. Dropout
model.add(Dropout(dropout_rate))

# 4. Couche Dense de sortie
model.add(Dense(output_classes, activation='softmax'))
```

5. Compilation de modèle

La compilation est une étape clé avant l'entraînement d'un modèle en Deep Learning. Elle permet de préparer le modèle à l'apprentissage en lui précisant :

Les éléments définis lors de la compilation :

1. **loss (fonction de perte)**
    - C'est la fonction que le modèle va chercher à **minimiser** pendant l'entraînement.
    - sparse_categorical_crossentropy est utilisée ici car :
        - Je fais une **classification multi-classes** (plus de deux classes).
        - Et les étiquettes sont **encodées sous forme d'entiers**, pas en one-hot.
2. **optimizer (optimiseur)**
    - L'algorithme utilisé pour **mettre à jour les poids** du modèle.
    - adam est un optimiseur très populaire, **rapide et efficace** pour la plupart des tâches de NLP.
3. **metrics (métriques d'évaluation)**
    - Ici, on surveille la **précision (accuracy)** pendant l'entraînement et l'évaluation du modèle.

```python
# Compilation du modèle
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

6. Entrainement de modèle

```python
# Paramètres d'entraînement
epochs = 10
batch_size = 32
validation_split = 0.2


history = model.fit(
    X_train,
    y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_split=validation_split,
    verbose=1
)
```

Dans cette étape, **j'ai entraîné mon modèle** à l'aide des données d'apprentissage X_train et y_train. J'ai utilisé la méthode model.fit() en définissant les paramètres suivants :

- **epochs = 10** :
  Le modèle passe **10 fois** sur l'ensemble des données d'apprentissage pour apprendre les motifs.
- **batch_size = 32** :
  Les données sont divisées en **lots de 32 échantillons** pour être traitées à la fois. Cela permet d'**optimiser la mémoire** et la vitesse d'entraînement.
- **validation_split = 0.2** :
  J'ai réservé **20 % des données d'entraînement** pour la validation interne du modèle. Cela permet de **suivre les performances** pendant l'entraînement sans toucher aux données de test.
- **verbose = 1** :
  Cela affiche les détails de l'entraînement **en temps réel** à chaque époque (loss, accuracy, etc.).

Le résultat de l'entraînement est stocké dans la variable history, qui me permettra ensuite de **visualiser l'évolution de la perte et de la précision**.

7. Évaluation du modèle

Après l'entraînement, j'ai évalué les performances du modèle sur les données de test (X_test, y_test). Cette étape est essentielle pour vérifier la capacité du modèle à généraliser à de nouvelles données jamais vues pendant l'entraînement.

```
[1]  loss, accuracy = model.evaluate(X_test, y_test, verbose=1)
     print(f"Test Loss: {loss:.4f}")
     print(f"Test Accuracy: {accuracy:.4f}")

     195/195 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.8609 - loss: 0.4950
     Test Loss: 0.4725
     Test Accuracy: 0.8691
```

Interprétation des résultats d'évaluation

L'évaluation du modèle sur les données de test a donné les résultats suivants :

- **Loss (perte) :** 0.4725
- **Accuracy (précision) :** 0.8691, soit environ **86,9%**

Ces résultats indiquent que :

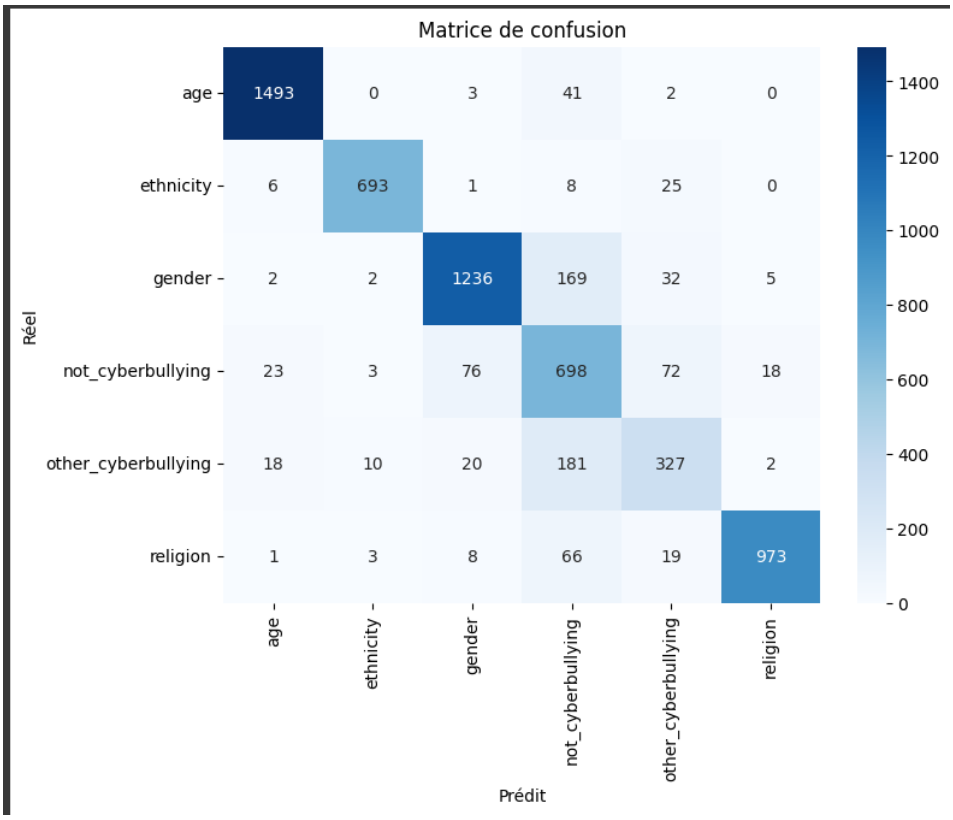
- Le modèle **parvient à classer correctement près de 87% des tweets** selon leur type de cyberharcèlement.
- La **valeur de la perte est relativement basse** (0.4725), ce qui montre que les erreurs de prédiction sont limitées.

Évaluation des performances par les scores de classification

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

```
                    precision    recall  f1-score   support

               age       0.97      0.97      0.97      1539
         ethnicity       0.97      0.95      0.96       733
            gender       0.92      0.85      0.89      1446
  not_cyberbullying      0.60      0.78      0.68       890
other_cyberbullying      0.69      0.59      0.63       558
          religion       0.97      0.91      0.94      1070

          accuracy                           0.87      6236
         macro avg       0.85      0.84      0.84      6236
      weighted avg       0.88      0.87      0.87      6236
```

Matrice de confusion



J'ai utilisé la matrice de confusion pour évaluer les performances de mon modèle de classification. Elle m'a permis de visualiser, pour chaque classe de cyberharcèlement, comment les prédictions se sont réparties par rapport aux valeurs réelles.

Points positifs que j'ai observés :

- Pour la classe **"age"**, le modèle a très bien performé avec **1493 prédictions correctes**, montrant une excellente précision.
- La classe **"gender"** a également donné de bons résultats, avec **1236 exemples bien classés**, même si **169 cas** ont été confondus avec "not_cyberbullying".
- Concernant **"religion"**, j'ai obtenu **973 bonnes prédictions**, ce qui témoigne d'une bonne capacité de discrimination du modèle.

Confusions que j'ai constatées :

- Le modèle confond parfois la classe **"gender"** avec **"not_cyberbullying"** (169 cas), ce qui indique une difficulté à faire la distinction dans certains tweets.
- J'ai remarqué que des tweets classés comme **"not_cyberbullying"** ont parfois été identifiés à tort comme du cyberharcèlement (et vice versa), ce qui pourrait générer des **faux positifs** dans un usage réel.

- Enfin, la classe **"other_cyberbullying"** semble poser plus de problèmes au modèle, avec plusieurs confusions avec d'autres classes, notamment "gender" et "not_cyberbullying". Cela pourrait s'expliquer par la nature plus variée ou moins représentée de cette catégorie dans les données.

8. Application du modèle sur de nouveaux messages textuels

8-1. Pré-traitement de nouveau message

```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()

# Nettoyage d'un seul tweet
def nettoyer_texte(tweet):
    tweet = tweet.lower()                                  # minuscules
    tweet = re.sub(r"http\S+|www\S+|https\S+", "", tweet)  # URL
    tweet = re.sub(r"@\w+", "", tweet)                     # mentions
    tweet = re.sub(r"\d+", "", tweet)                      # nombres
    tweet = re.sub(r"[^\w\s]", "", tweet)                  # ponctuation
    tweet = re.sub(r"[#@!%?\*&$€+=\[\]\(\)\{\}<>\\/:;\"'\|]", "", tweet)  # caractères spéciaux
    tokens = tweet.split()                                 # découper en mots
    tokens = [w for w in tokens if w not in stop_words] # retirer stop words
    tokens = [stemmer.stem(w) for w in tokens]             # stemming
    return " ".join(tokens)
```

8-2. Prédiction de la catégorie d'un nouveau message

```python
def predire_tweet(nouveau_tweet, tokenizer, model, label_encoder, maxlen):
    tweet_nettoye = nettoyer_texte(nouveau_tweet)

    # Tokenisation (transformation en indices avec le tokenizer déjà entraîné)
    sequence = tokenizer.texts_to_sequences([tweet_nettoye])

    # Padding
    sequence_pad = pad_sequences(sequence, maxlen=maxlen, padding='post')

    # Prédiction
    prediction = model.predict(sequence_pad)
    classe_predite = np.argmax(prediction, axis=1)

    # Décodage du label
    label = label_encoder.inverse_transform(classe_predite)
    return label[0]
```

```python
nouveau_tweet = "isi account pretend kurdish account like islam"
classe = predire_tweet(nouveau_tweet, tokenizer, model, label_encoder, maxlen=50)
print("Classe prédite :", classe)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
Classe prédite : other_cyberbullying
```

9. Enregistrement du modèle pour une réutilisation

```
model.save("cyberbullying_model.keras")
```

10. Intégration et exploitation du modèle entraîné

10-1. Création d'une API Flask pour l'inférence du modèle

Installation des dépendances nécessaires

Pour exécuter l'API Flask dans un environnement comme Google Colab, j'ai installé les bibliothèques suivantes :

```
!pip install pyngrok
!pip install flask-ngrok
```

Déploiement de l'API Flask pour la prédiction

Dans cette étape, j'ai développé une API avec **Flask** pour permettre à des applications externes (comme une plateforme sociale) d'envoyer un **texte** (tweet ou commentaire) et recevoir une **prédiction** du modèle entraîné.

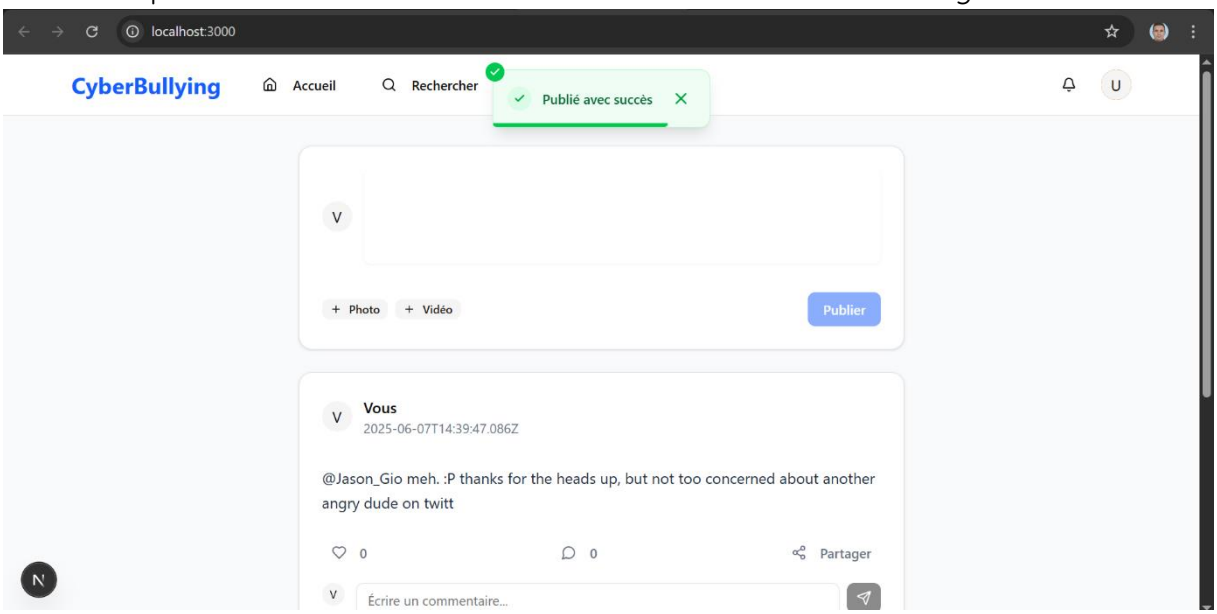
- J'ai utilisé **Ngrok** pour exposer localement l'API à Internet via une URL publique.
- Le modèle entraîné (cyberbullying_model.keras), le **tokenizer** et le **label encoder** sont chargés depuis Google Drive.
- Une fonction preprocess() est utilisée pour préparer les textes reçus via l'API.
- Le point de terminaison /predict reçoit une requête **POST** contenant un texte, applique le modèle, et retourne la classe prédite (type de cyberbullying ou "non cyberbullying").

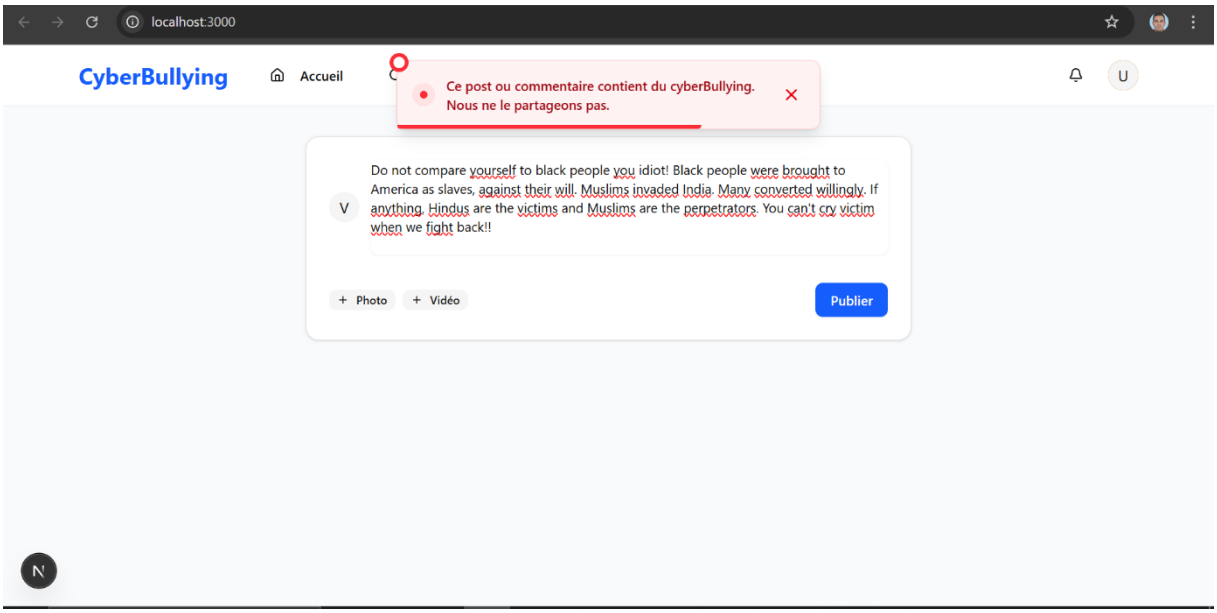
```
from typing import Text
from flask import Flask, request, jsonify
from pyngrok import ngrok
import numpy as np
from keras.models import load_model
import pickle
from keras.preprocessing.sequence import pad_sequences

ngrok.set_auth_token("votre_cle_ngrok")
public_url = ngrok.connect(5000)
print("ngrok tunnel URL:", public_url)

model = load_model('/content/drive/MyDrive/cyberbullying_model.keras')
with open('/content/drive/MyDrive/tokenizer.pkl', 'rb') as f:
    tokenizer = pickle.load(f)
with open('/content/drive/MyDrive/label_encoder.pkl', 'rb') as f:
    label_encoder = pickle.load(f)

def preprocess(text):
    print("texte avant nettoyage  ",text)
    text =nettoyer_texte(text)
    print("texte nettoyer est :",text)
    seq = tokenizer.texts_to_sequences([text])
    padded = pad_sequences(seq, maxlen=100)
    return padded

app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    text = data["text"]
    processed = preprocess(text)
    pred = model.predict(processed)

    class_id = np.argmax(pred,axis=1)
    class_label = label_encoder.inverse_transform(class_id)[0]
    return jsonify({'prediction': class_label})
app.run()
```

10-2. Connexion de l'API Flask à un site front-end Next.js

Dans cette étape, j'ai développé un mini site web permettant aux utilisateurs de publier des posts textuels et d'ajouter des commentaires. Avant la publication d'un post, une prédiction est effectuée pour déterminer s'il s'agit de cyberharcèlement. Si le contenu n'est pas considéré comme du cyberharcèlement, le post est publié. En revanche, si un contenu offensant est détecté, un message d'erreur s'affiche pour informer l'utilisateur que ce type de post n'est pas autorisé sur notre site.

Les images ci-dessous montrent les principales fonctionnalités de l'application, notamment la publication de posts, l'ajout de commentaires et la détection automatique de cyberharcèlement.





10-3 Etapes d'utilisation

Etape 1 : Exécution de l'application Flask dans le fichier **FLASKAPI** sur Google Colab. Un lien sera généré

```
ngrok tunnel URL: NgrokTunnel: "https://58ad-34-125-168-22.ngrok-free.app" -> "http://localhost:5000"
```

Etape 2 : copie le lien https://58ad-34-125-168-22.ngrok-free.app

Etape 3 : remplacer ce lien dans nextjs exactement dans app components socialplateforme : const url="https://385f-34-170-210-143.ngrok-free.app/predict" //remplacer par l'url copie sans supprimer /predict.

Etape 4 : lancer backend avec **py app.py** est une petite backend pour CRUD des postes dans la base de données.

Etape 5 : Lancer frontend avec npm run dev.

11. Limites et Contraintes

Durant la réalisation de ce projet, plusieurs contraintes ont été rencontrées :

- **Limitations matérielles :** l'entraînement du modèle s'est effectué sur un processeur (CPU), ce qui a considérablement ralenti le processus d'apprentissage et allongé le temps de traitement, surtout avec des données volumineuses.
- **Qualité et diversité des données** : le jeu de données utilisé contenait des tweets rédigés en plusieurs langues, ce qui a pu nuire à la cohérence de l'apprentissage. De plus, la quantité globale de données était relativement limitée pour un modèle de Deep Learning, ce qui peut impacter la généralisation et la robustesse du modèle.

12. Conclusion

Ce projet a permis de mettre en place un système de classification automatique des tweets et commentaires en lien avec le cyberharcèlement, en s'appuyant sur des techniques de traitement du langage naturel et un modèle de deep learning. Grâce à un prétraitement rigoureux et une architecture adaptée, le modèle est capable de détecter différentes formes de cyberintimidation, contribuant ainsi à modérer efficacement les contenus publiés sur une plateforme sociale.

Malgré certaines limitations techniques et liées aux données, les résultats obtenus sont satisfaisants et ouvrent la voie à des améliorations futures, telles que l'extension du dataset, l'utilisation de modèles plus avancés et l'intégration complète dans une application web fonctionnelle.