



## **NLP – Modération de Chat**

Nakache Eithan

Ziane Camil

Hadj-Said Samy

Perez Jason

Six Briac

Bellamy Baptiste

### **Sujet**

Modération en temps réel des chats sur une plateforme de contenu, par l'intermédiaire de  
modèle NLP

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Présentation du jeu de données</b>	<b>2</b>
<b>3</b>	<b>Statistiques descriptives du jeu de données</b>	<b>4</b>
<b>4</b>	<b>Tokenizer</b>	<b>6</b>
4.1	Prétraitement du jeu de données . . . . .	6
4.2	Comparaison avec d'autres <i>tokenizers</i> . . . . .	6
4.3	Méthodes de normalisation du texte . . . . .	7
4.4	Prétraitement supplémentaire du jeu de données . . . . .	8
<b>5</b>	<b>Naive Bayes</b>	<b>11</b>
5.1	Pré-traitement des données . . . . .	11
5.1.1	Tokenisation des commentaires . . . . .	11
5.2	Analyse exploratoire des données . . . . .	11
5.2.1	Statistiques descriptives . . . . .	11
5.3	Entraînement du modèle . . . . .	11
5.3.1	Préparation des données . . . . .	11
5.3.2	Évaluation des performances du modèle . . . . .	11
5.3.3	Validation croisée . . . . .	12
5.4	Analyse des résultats . . . . .	12
5.4.1	Calcul des probabilités . . . . .	12
5.4.2	Calcul des vraisemblances . . . . .	12
5.4.3	Prédiction . . . . .	12
5.4.4	Matrice de confusion . . . . .	12
5.4.5	Etudes Annexe sur les données . . . . .	13
<b>6</b>	<b>N-gram</b>	<b>15</b>
6.1	Pré-traitement des données . . . . .	15
6.1.1	Tokenisation des commentaires . . . . .	15
6.2	Analyse exploratoire des données . . . . .	15
6.2.1	Statistiques descriptives . . . . .	15
6.3	Entraînement du modèle . . . . .	15
6.3.1	Préparation des données . . . . .	15
6.3.2	Création du modèle de bigrams et trigrams . . . . .	16
6.3.3	Fréquence des n-grams . . . . .	16
6.3.4	Fonction de prédiction . . . . .	16
6.4	Évaluation des performances du modèle . . . . .	17

6.5	Analyse des résultats . . . . .	17
6.5.1	Interprétation des fréquences des n-grams . . . . .	17
6.5.2	Prédiction et complétion de phrases . . . . .	17
6.6	Conclusion . . . . .	17
<b>7</b>	<b>Régression Logistique</b>	<b>18</b>
7.1	TF-IDF . . . . .	18
7.2	Word2Vec . . . . .	20
7.3	Régression Multilabel . . . . .	22
<b>8</b>	<b>Feed Forward Neural Networks</b>	<b>23</b>
8.1	Étape de la construction du modèle . . . . .	23
8.2	Les embeddings . . . . .	23
8.2.1	GloVe . . . . .	23
8.2.2	FastText . . . . .	23
8.3	Architecture du modèle . . . . .	23
8.3.1	Fonction de coût . . . . .	24
8.3.2	Optimisation . . . . .	24
8.4	Création des modèles . . . . .	24
8.5	Résultats . . . . .	24
<b>9</b>	<b>Transformers</b>	<b>26</b>
9.1	Construction du modèle . . . . .	26
9.1.1	Prétraitement des données . . . . .	26
9.1.2	Architecture du modèle . . . . .	26
9.2	Résultats . . . . .	26
<b>10</b>	<b>Limitation et Perspectives</b>	<b>28</b>
10.1	Limitations . . . . .	28
10.2	Perspectives . . . . .	28
<b>A</b>	<b>Appendix</b>	<b>29</b>
<b>B</b>	<b>Appendix</b>	<b>30</b>
B.1	Corrélation entre les labels . . . . .	30
B.2	Distribution des mots . . . . .	31
<b>C</b>	<b>Appendix</b>	<b>32</b>
C.1	Word Cloud Faux Positifs . . . . .	32
C.2	Word Cloud Faux Négatifs . . . . .	33

C.3 Shap Values . . . . .	34
---------------------------	----

## 1. Introduction

Ce projet vise à développer des modèles de traitement du langage naturel dédiés à la modération en temps réel des chats sur une plateforme de contenu. L'objectif principal est d'assurer la modération des messages des utilisateurs afin de prévenir la diffusion de contenus inappropriés, tels que les insultes, les spams et autres formes de communications indésirables.

### Dataset

Le dataset choisi pour ce projet est le **Jigsaw Toxic Comment Classification**. Ce dataset recense un grand nombre de commentaires anglais, provenant de *Wikipedia*, labellisés par des humains en fonction de leur toxicité. Ce dernier provient d'une compétition *Kaggle* et peut être obtenu à partir du lien suivant : [jigsaw toxic comment classification challenge](#).

### Objectif

Plusieurs modèles de *machine learning* et de *deep learning* seront développés et entraînés dans le but de prédire la pertinence et l'acceptabilité des messages. Une fois ces modèles développés, nous procéderons à une évaluation rigoureuse de leurs performances par comparaison mutuelle. La dernière étape de ce projet consistera en la création d'une preuve de concept sous la forme d'une interface web.

Cette interface permettra de modérer en direct les messages échangés sur un chat de *live streaming*, démontrant ainsi l'applicabilité pratique de nos recherches.

## 2. Présentation du jeu de données

### Objectifs et financement du dataset

Le jeu de données de classification des commentaires toxiques de *Jigsaw* a été créé pour promouvoir la recherche sur la détection de la toxicité dans les commentaires en ligne. Il vise à identifier des comportements indésirables tels que les commentaires toxiques. Ce corpus a été collecté dans le but de développer des technologies et des méthodes capables de modérer automatiquement ces types de contenu nuisible sur les plateformes en ligne.

Le projet a été financé par *Jigsaw* (anciennement connu sous le nom de *Google Ideas*) et *Google*, dans le cadre d'un concours organisé sur la plateforme *Kaggle*. Ce partenariat a non seulement mis à disposition les ressources nécessaires, mais a également encouragé la communauté globale des *data scientists* à résoudre ce problème urgent de modération des contenus toxiques sur Internet.

### Contexte et caractéristiques des données du dataset

Les commentaires inclus dans le jeu de données proviennent des pages de discussion de *Wikipedia*. Ces discussions sont menées en anglais par des contributeurs qui échangent sur les améliorations à apporter aux articles et sur les modifications nécessaires. Ces échanges sont caractéristiques des interactions collaboratives typiques sur *Wikipedia*, où les utilisateurs débattent de la véracité, de la neutralité et de la complétude des articles.

Le format du texte est informel mais structuré, similaire aux communications en ligne, et peut contenir des fautes, des abréviations, ou des mots allongés ou raccourcis. Cela signifie que les commentaires, bien que rédigés dans un cadre informel, suivent une certaine structure logique et sont orientés vers des objectifs spécifiques de collaboration et de modification de contenu.

### Démographie des auteurs

Les informations démographiques spécifiques sur les auteurs des commentaires ne sont pas fournies.

### Processus de collecte

Le jeu de données comprend environ **160 000** commentaires pour l’entraînement et **60 000** commentaires pour le test. Ces données ont été extraites dans le but de représenter divers comportements toxiques, bien que la méthode exacte d’échantillonnage n’ait pas été spécifiée.

Étant issues de plateformes ouvertes, les questions de consentement sont gérées dans le cadre des normes de *Wikipedia* concernant la publication de commentaires publics. Toutefois, les détails spécifiques concernant le consentement des auteurs ne sont pas divulgués.

En ce qui concerne le prétraitement, les données ont été anonymisées et les informations personnellement identifiables ont été supprimées pour protéger la vie privée des utilisateurs.

### Processus d’annotation

Le jeu de données est structuré autour de plusieurs catégories d’annotations qui permettent de définir la nature de la toxicité des commentaires. Celles-ci incluent : **Toxique**, **Très toxique**, **Obscène**, **Menace**, **Insulte**, **Haine identitaire**.

La méthode d’annotation repose sur l’intervention de multiples annotateurs pour chaque commentaire. Le recours à plusieurs annotateurs permet de réduire les biais individuels et d’améliorer la précision générale des données annotées, assurant ainsi que les modèles entraînés avec ce jeu de données peuvent fonctionner de manière efficace et équitable.

### Distribution

Le jeu de données est disponible à des fins de recherche non commerciales. Les utilisateurs doivent généralement accepter des conditions d’utilisation qui limitent l’utilisation commerciale et la redistribution.

### Erreur et Biais

La labellisation des commentaires est sujette à des biais humains et des erreurs de classification. Dans le jeu de test, il y a une partie non négligeable de phrases classées non toxiques qui le sont en réalité. ([[van Aken et al.\(2018\)](#)[van Aken, Risch, Krestel, and Loser](#)] )

### 3. Statistiques descriptives du jeu de données

#### Division du dataset

Le jeu de données a été divisé en trois parties : entraînement, validation et test. Voici la répartition du nombre de commentaires dans chacune de ces parties :

Catégorie	Nombre de commentaires
Train	127,656
Validation	31,915
Test	63,978

TABLE 3.1 – Répartition du nombre de commentaires

#### Répartition des labels

Les commentaires toxiques sont minoritaires dans l'ensemble de données. En effet, il y a **10.2%** de commentaire globalement non toxiques. Cela peut poser des problèmes lors de l'entraînement des modèles, car les classes minoritaires peuvent être sous-représentées et donc mal apprises. La répartition des labels de toxicité est également inégale. La somme des pourcentages n'est pas égale à 100% car un commentaire peut avoir plusieurs labels. C'est donc un problème de classification *multi labels*.

Label	Pourcentage
toxic	94.3%
severe_toxic	9.8%
obscene	51.9%
threat	3.1%
insult	48.2%
identity-hate	8.6%

TABLE 3.2 – Répartition des labels sur les commentaires globalement toxiques



## Corrélation entre les labels

Les labels de toxicité présentent une forte corrélation entre eux. Il est possible d'anticiper dès à présent une difficulté du modèle à distinguer un commentaire toxique d'un commentaire obscène (**74%** de corrélation). Cela représente un point à prendre en compte lors de la conception du modèle. La matrice de corrélation est incluse dans l'appendice B.

## Longueur des commentaires

La distribution de la longueur des commentaires est très variée. En effet, l'écart-type est bien plus élevé que la moyenne. La longueur moyenne des commentaires est de **395** caractères, avec un écart-type de **593** caractères. Cela peut poser des défis lors de la conception du modèle. Il est crucial de prétraiter les données pour normaliser la longueur des commentaires.

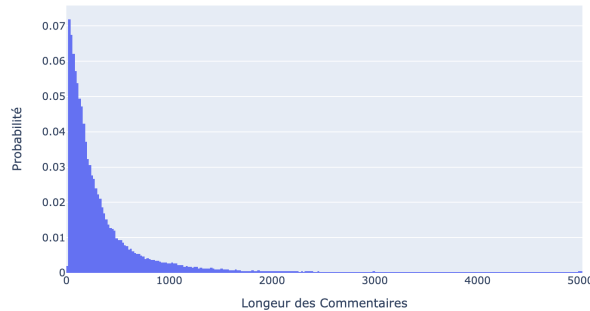


FIGURE 3.1 – Distribution de la longueur des commentaires

## Distribution des mots

Dans le jeu de données, environ **180 000** mots uniques sont recensés. Sans surprise, les mots les plus fréquents sont les mots de liaison et les mots vides. Cependant, en raison de la source du jeu de données, les termes **articles**, **wikipedia** et **pages** apparaissent également très souvent dans les commentaires (respectivement en 26e, 30e et 31e positions). En effet, les utilisateurs citent souvent des sources pour appuyer leurs propos. Dans les commentaires toxiques, des insultes, des mots vulgaires et des termes discriminatoires sont fréquemment observés. La visualisation de ces derniers à l'aide de *WordCloud* permet de représenter visuellement les mots les plus utilisés. En annexe B, une représentation *WordCloud* des commentaires du jeu de données, filtrée selon le type de toxicité est présentée.

## 4. Tokenizer

### 4.1 Prétraitement du jeu de données

Le prétraitement des données est une étape essentielle dans le processus d'analyse de texte. Il vise à nettoyer, transformer et préparer les données brutes afin de les rendre exploitables pour l'entraînement des modèles. Dans cette section, des techniques de prétraitement seront appliquées sur notre jeu de données afin de le rendre apte à être utilisé dans nos modèles prédictifs.

#### Tokenisation à base d'expressions régulières (*RegexTokenizer*)

La tokenisation est le processus de division du texte en unités plus petites appelées "tokens". La tokenisation à base d'expressions régulières utilise des règles basées sur des motifs d'expressions régulières pour diviser le texte en tokens.

Il convient de commencer par importer la classe *RegexTokenizer* et d'appliquer la tokenisation sur le jeu de données.

Avant tokenisation	Après tokenisation
"Hello, World!"	['Hello', ',', 'World', '!']
"I love NLTK"	['I', 'love', 'NLTK']

TABLE 4.1 – Exemple de tokenisation à base d'expressions régulières

#### Tokenisation byte-pair encoding (*TikToken*)

Le *Byte-Pair Encoding (BPE)* est une méthode de tokenisation qui découpe le texte en sous-unités de texte appelées "tokens" en utilisant un algorithme de compression de données.

La tokenisation BPE est appliquée sur le jeu de données en utilisant la bibliothèque *TikToken*.

Avant tokenisation	Après tokenisation
"Hello, World!"	['Hello', ',', 'World', '!']
"I love <i>TikToken</i> "	['I', 'love', 'Ti', 'k', 'To', 'ken']

TABLE 4.2 – Exemple de tokenisation BPE

### 4.2 Comparaison avec d'autres *tokenizers*

Une comparaison a également été effectuée sur la performance de notre *tokenizer* avec d'autres options disponibles dans la bibliothèque MinBPE.

Tokenizer	Temps d'entraînement (minutes)
RegexTokenizer	182
<i>TikToken</i>	0 (déjà entraîné)
BasicTokenizer	69

TABLE 4.3 – Comparaison des temps d'entraînement des *tokenizers*

Il est constaté que le *RegexTokenizer* et le *TikToken* sont significativement plus rapides que le *BasicTokenizer*. *TikToken* est le plus rapide de tous les *tokenizers* testés.

### 4.3 Méthodes de normalisation du texte

La normalisation du texte est une étape cruciale du prétraitement des données textuelles. Elle vise à uniformiser le texte en le mettant en minuscules, en supprimant les *stop words* et en lemmatisant les mots.

#### Suppression des *stop words*

Les *stop words* sont des mots courants qui n'apportent pas beaucoup de valeur sémantique au texte (par exemple : '*a*', '*the*', '*for*', ...). Ces éléments seront supprimés du jeu de données.

#### Lemmatisation

La lemmatisation consiste à réduire les mots fléchis ou dérivés à leur forme de base ou racine. Cela permet de normaliser le texte et de réduire la dimensionnalité de l'espace des *features*.

#### Mise en minuscules

La mise en minuscules permet d'uniformiser le texte en convertissant toutes les lettres en minuscules. Cela permet d'éviter les doublons dus à la casse.

#### Identification des verbes et des noms

La méthode de lemmatisation a été modifiée pour identifier les verbes et les noms dans une phrase. Parfois, les verbes étaient considérés comme des noms et ne pouvaient pas être lemmatisés avec le lemmatiseur par défaut.

#### Les caractères spéciaux

En examinant attentivement notre base de données, il a été constaté la présence de nombreux tokens représentant des *URL*, des adresses e-mail et d'autres éléments inutiles pour nos

calculs d'entraînement des modèles. Par conséquent, une décision a été prise de les supprimer afin de réduire au maximum le bruit dans nos données et ainsi optimiser nos modèles.

### Remplacement des emojis

Certains emojis peuvent être importants pour l'indication d'une toxicité ou non c'est pourquoi il a été décidé de remplacer certains emojis par un mot les représentant au mieux afin de permettre aux modèles de mieux s'adapter.

### Effacement des ponctuations

Parfois, certaines ponctuations occupent trop d'espace sans être utiles au modèle. Ainsi, tous les caractères qui ne sont pas dans la table *ASCII* et les signes de ponctuation qui pourraient compromettre l'entraînement des modèles seront supprimées.

### Effacement des duplications

En constatant que plusieurs messages vulgaires étaient dissimulés parmi les *tokens* en raison de la duplication des premières et/ou dernières lettres, il a été nécessaire de corriger ces *tokens* pour leur attribuer une plus grande valeur.

### Tableau représentatif des différentes fonctions de prétraitement

Le tableau représentatif des différentes fonctions de prétraitement peut être consulté dans l'annexe dédiée au *tokenizer* (Annexe A).

## 4.4 Prétraitement supplémentaire du jeu de données

En plus des techniques de prétraitement déjà mentionnées, les auteurs ont également effectué les actions suivantes pour rendre le jeu de données plus adapté à l'entraînement des modèles :

### Réflexion sur la suppression des symboles opérateurs et des chiffres

Dans le cadre du prétraitement des données, il avait été envisagé initialement de supprimer les symboles opérateurs et les chiffres présents dans les textes. Cependant, après réflexion, il a été réalisé que ces symboles pouvaient avoir une valeur sémantique importante pour déterminer si une phrase est toxique ou non. Ainsi, au lieu de les supprimer, un processus sophistiqué a été mis en place consistant à utiliser une série d'expressions régulières (regex) pour remplacer ces symboles par leur signification correspondante. Par exemple, un remplacement a été effectué sur des symboles comme " :/" par leur signification afin d'améliorer la compréhension du texte par nos modèles. Cette approche permet d'assurer des interactions plus contextuelles

et pertinentes avec les utilisateurs, tout en garantissant une analyse précise et efficace du langage. Chacune de ces expressions régulières a été soigneusement personnalisée par notre équipe pour répondre aux besoins spécifiques de notre projet, permettant ainsi à notre *IA* de mieux comprendre le langage naturel et d'interagir de manière plus fluide avec les utilisateurs.

Élément toxique	Fréquence
8=====d	penis
:))))	happy
:(	sad
:)	happy
:D	laugh
8=====>	penis

TABLE 4.4 – Exemple d'éléments toxiques contenant des symboles opérateurs et des chiffres qui ont été remplacés par leur signification.

### Suppression des balises *HTML*, des *URL* et des adresses e-mail

Les balises *HTML*, les *URL* et les adresses e-mail ont été supprimées du jeu de données. Ces éléments étaient fréquents dans notre base de données en raison d'erreurs de récupération de données lors de la collecte de commentaires sur *Wikipedia*.

### Correction des erreurs orthographiques et des fautes de frappe

Un algorithme de correction des erreurs orthographiques et des fautes de frappe a été utilisé pour améliorer la qualité des données textuelles. Cet algorithme remplace les mots incorrectement orthographiés par les mots les plus proches dans notre dictionnaire de référence. Par exemple les caractères spéciaux, tels que 'i' utilisé à la place de 'i', seront corrigés. Voici des exemples de phrases avant et après l'application de notre algorithme de correction :

Avant correction	Après correction
lies	lies
penisd i'\m	penis i'm
fück8	fuck

TABLE 4.5 – Exemple de phrase avant et après correction

### Non-application de la correction d'orthographe

Initialement, l'équipe avait envisagé d'appliquer un algorithme de correction d'orthographe pour améliorer la qualité des données textuelles. Cependant, après avoir constaté que cette étape prenait trop de temps et pouvait introduire des erreurs, notamment en modifiant des noms propres en mots n'ayant aucun rapport, la décision a été prise de ne pas l'appliquer. En effet, cela pourrait introduire des biais importants, notamment si un prénom ressemble à une insulte, ce qui pourrait transformer une phrase anodine en une phrase insultante. Par conséquent, il a été choisi de ne pas effectuer cette étape de correction d'orthographe pour garantir l'intégrité et la qualité des données textuelles.

### Exemple d'une phrase après toutes les étapes de normalisation

Étape	Texte normalisé
Texte initial	motherrrrrrrrrrrFuckkkkkkkk!!!aaaaaa***** stringsss ***!!!!!!bitchhhhhh :) 8=====
Après remplacement des emojis	motherrrrrrrrrrrFuckkkkkkkk!!!aaaaaa***** stringsss ***!!!!!!bitchhhhhh happy 8=====
Après suppression des caractères spéciaux	motherFuck a strings bitch happy penis
Après tokenisation	'motherFuck', 'a', 'strings', 'bitch', 'happy', 'penis'
Après conversion en minuscules	'motherfuck', 'a', 'strings', 'bitch', 'happy', 'penis'
Après lemmatisation	'motherfuck', 'a', 'string', 'bitch', 'happy', 'penis'
Après suppression des stopwords	'motherfuck', 'string', 'bitch', 'happy', 'penis'
Texte final après retokenisation	mother fuck string bitch happy penis

TABLE 4.6 – Étapes de normalisation du texte

## 5. Naive Bayes

### 5.1 Pré-traitement des données

Nous avons d'abord effectué un pré-traitement des données en utilisant différentes techniques :

#### 5.1.1 Tokenisation des commentaires

Nous avons utilisé la tokenisation pour diviser les commentaires en tokens individuels.

#### Tokenisation à l'aide de notre API `tokenize_apy`

La tokenisation des commentaires a été réalisée en utilisant l'API `tokenize_apy`.

#### Équilibrage du jeu de données

Nous avons équilibré le jeu de données en échantillonnant un nombre égal de commentaires toxiques et non toxiques.

### 5.2 Analyse exploratoire des données

Voici les statistiques descriptives sur le jeu de données équilibré :

#### 5.2.1 Statistiques descriptives

- Nombre total de documents : 25 962
- Nombre total de tokens : 127 656
- Nombre de classes à prédire : 2 (toxique ou non toxique)
- Les tokens les plus fréquents incluent "grand", "ter", "burn", etc.

### 5.3 Entraînement du modèle

Nous avons entraîné le modèle Bayésien naïf en suivant les étapes suivantes :

#### 5.3.1 Préparation des données

- Calcul de la fréquence des mots et sélection des stop words.
- Entraînement du modèle en utilisant `CountVectorizer` et `MultinomialNB`.

#### 5.3.2 Évaluation des performances du modèle

Les "Feature Dimensions" indiquent la taille de la matrice de caractéristiques utilisée pour chaque méthode. Le premier nombre représente le nombre d'échantillons (lignes) dans l'ensemble de données, et le deuxième nombre représente le nombre de caractéristiques (colonnes) générées par le vectoriseur.

TABLE 5.1 – Rapports de classification combinés pour différentes méthodes

Méthode	Dimensions des Features	Classe 0		Classe 1		Précision	Macro Moy.	Moy. Pondérée
		Précision	Rappel	Précision	Rappel			
comment_text_baseline	(63978, 52493)	0.98	0.85	0.39	0.86	0.85	(0.68, 0.86, 0.72)	(0.92, 0.85, 0.88)
comment_text_bpe_tokenize_full_normalization	(63978, 22188)	0.98	0.86	0.40	0.87	0.86	(0.69, 0.86, 0.73)	(0.93, 0.86, 0.88)
comment_text_word_tokenize_no_normalization	(63978, 52478)	0.98	0.85	0.39	0.86	0.85	(0.68, 0.85, 0.72)	(0.92, 0.85, 0.88)
comment_text_gpt_tokenize_no_normalization	(63978, 44632)	0.97	0.90	0.45	0.74	0.89	(0.71, 0.82, 0.75)	(0.92, 0.89, 0.90)
comment_text_word_tokenize_normalization	(63978, 42934)	0.98	0.85	0.39	0.87	0.85	(0.69, 0.86, 0.73)	(0.93, 0.85, 0.88)
comment_text_gpt_tokenize_normalization	(63978, 24578)	0.98	0.87	0.40	0.80	0.86	(0.69, 0.84, 0.73)	(0.92, 0.86, 0.88)
comment_text_word_tokenize_full_normalization	(63978, 41908)	0.98	0.85	0.39	0.88	0.85	(0.69, 0.86, 0.73)	(0.93, 0.85, 0.88)
comment_text_gpt_tokenize_full_normalization	(63978, 22750)	0.98	0.86	0.40	0.85	0.86	(0.69, 0.85, 0.73)	(0.92, 0.86, 0.88)
comment_text_word_tokenize_simple_normalization	(63978, 48335)	0.98	0.85	0.39	0.86	0.85	(0.68, 0.86, 0.72)	(0.92, 0.85, 0.88)
comment_text_gpt_tokenize_simple_normalization	(63978, 29389)	0.98	0.86	0.39	0.82	0.86	(0.69, 0.84, 0.72)	(0.92, 0.86, 0.88)
comment_text_bpe_tokenize_simple_dup_normalization	(63978, 27126)	0.98	0.86	0.39	0.85	0.86	(0.69, 0.85, 0.73)	(0.92, 0.86, 0.88)
comment_text_bpe_tokenize_no_dup_no_punc_normalization	(63978, 27290)	0.98	0.86	0.39	0.85	0.86	(0.69, 0.85, 0.73)	(0.92, 0.86, 0.88)

### 5.3.3 Validation croisée

Nous avons également effectué une validation croisée en utilisant la métrique `f1_macro`.

Métrique	Score
f1_macro (moyenne)	0.8438
f1_macro (écart type)	0.0055

TABLE 5.2 – Résultats de la validation croisée

## 5.4 Analyse des résultats

### 5.4.1 Calcul des probabilités

- Probabilités a priori pour chaque classe :  $\{0 : 0.89831, 1 : 0.10168\}$
- Vocabulaire contient 33397 mots uniques

### 5.4.2 Calcul des vraisemblances

Nous avons calculé les probabilités conditionnelles (vraisemblances) pour chaque mot dans chaque classe.

### 5.4.3 Prédiction

Nous avons utilisé le modèle entraîné pour prédire la toxicité des commentaires de test. Par exemple, le commentaire "chretien attack" a été prédit comme toxique avec une probabilité de 0.000168.

### 5.4.4 Matrice de confusion

La matrice de confusion montre que le modèle prédit correctement les commentaires toxiques et non toxiques dans une large mesure.



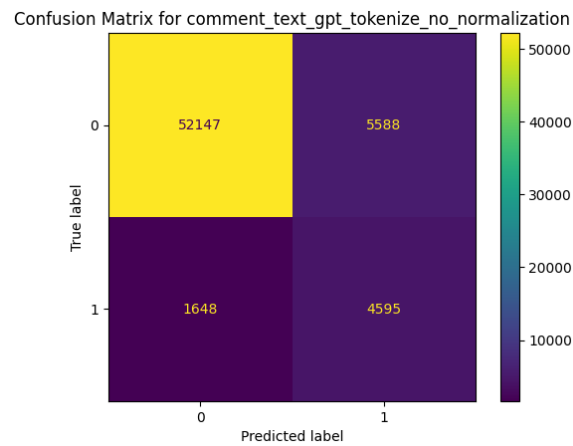


FIGURE 5.1 – Matrice de confusion du modèle Bayésien naïf

#### 5.4.5 Etudes Annexe sur les données

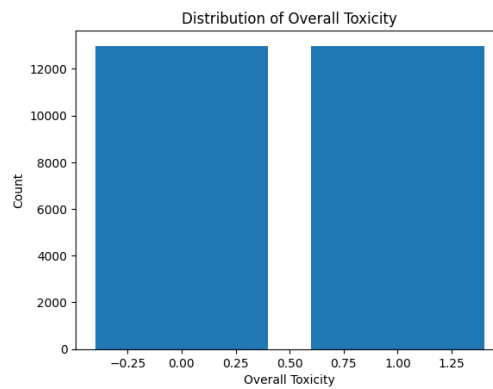


FIGURE 5.2 – Distribution de la toxicité des commentaires

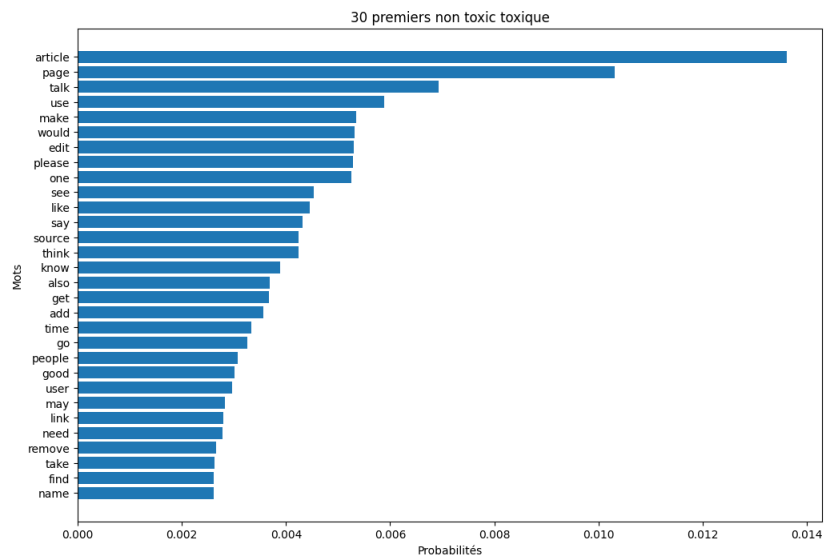


FIGURE 5.3 – Top 30 mots non toxiques

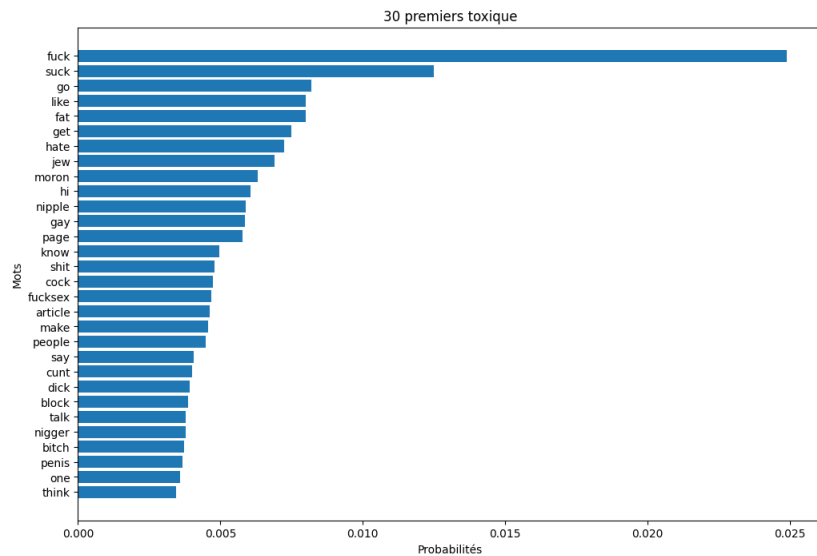


FIGURE 5.4 – Top 30 mots toxiques

## 6. N-gram

### 6.1 Pré-traitement des données

Nous avons d'abord effectué un pré-traitement des données en utilisant différentes techniques pour améliorer la qualité des entrées pour le modèle N-gram.

#### 6.1.1 Tokenisation des commentaires

La tokenisation divise les commentaires en unités de base appelées tokens, facilitant l'analyse et la modélisation des données textuelles.

##### Tokenisation à l'aide de notre API `tokenize_apy`

Nous avons utilisé l'API `tokenize_apy` pour effectuer la tokenisation des commentaires. Cette API nous permet de transformer les phrases en une séquence de mots individuels ou tokens, essentielle pour la création des n-grams.

#### Équilibrage du jeu de données

Pour éviter les biais dans notre modèle, nous avons équilibré le jeu de données en échantillonnant un nombre égal de commentaires toxiques et non toxiques. Cette étape est cruciale pour s'assurer que le modèle apprend de manière équitable des deux types de données.

### 6.2 Analyse exploratoire des données

Avant de procéder à l'entraînement du modèle, nous avons effectué une analyse exploratoire pour comprendre les caractéristiques de notre jeu de données.

#### 6.2.1 Statistiques descriptives

Voici les statistiques descriptives sur le jeu de données équilibré :

- Nombre total de documents : 223 549
- Nombre total de tokens : Variable selon les n-grams
- Les n-grams les plus fréquents incluent "nigger nigger", "on my", "be blocked", etc.

### 6.3 Entraînement du modèle

Nous avons entraîné notre modèle N-gram en suivant une série d'étapes méthodiques.

#### 6.3.1 Préparation des données

- Utilisation de `CountVectorizer` pour créer des bigrams, trigrams et 4-grams.
- Entraînement du modèle en utilisant les fréquences des n-grams extraits des commentaires.

### 6.3.2 Création du modèle de bigrams et trigrams

Nous avons utilisé ‘CountVectorizer’ pour créer des modèles de bigrams (séquences de deux mots) et trigrams (séquences de trois mots) à partir des phrases.

### 6.3.3 Fréquence des n-grams

Nous avons calculé la fréquence des n-grams les plus courants dans le jeu de données pour avoir une meilleure compréhension de leur distribution.

Bigram	Fréquence
nigger nigger	3411
on my	3067
be blocked	2761
fuck you	2681

TABLE 6.1 – Fréquence des bigrams les plus courants

Trigram	Fréquence
once upon a	26
on my talk	20
be blocked from	15

TABLE 6.2 – Fréquence des trigrams les plus courants

### 6.3.4 Fonction de prédiction

Nous avons défini des fonctions de prédiction pour compléter les phrases basées sur les n-grams les plus fréquents.

#### Prédiction avec les bigrams

— Exemple de prédiction bigram : "Do you like playing" → "the"

#### Prédiction avec les trigrams

— Exemple de prédiction trigram : "once upon a" → "time"

#### Complétion de phrase

Nous avons également créé une fonction `complete_the_phrase` qui prend en entrée une phrase de départ et prédit la suite.

— Exemple d'utilisation : "once upon" → "a time when I was just a few days ago I'm not"

## 6.4 Évaluation des performances du modèle

Pour évaluer les performances de notre modèle, nous avons calculé son accuracy. Nous avons testé le modèle sur une cinquantaine de phrases concaténées afin d'obtenir ces résultats.

N-gram	Accuracy
Bigram	0.0925
Trigram	0.0925

TABLE 6.3 – Performances du modèle N-gram avec différentes configurations (sur 50 phrases)

## 6.5 Analyse des résultats

### 6.5.1 Interprétation des fréquences des n-grams

Les n-grams les plus fréquents identifiés révèlent des schémas de langage courants et des expressions fréquentes dans le corpus. Ces informations sont cruciales pour améliorer la compréhension et la génération de texte.

### 6.5.2 Prédiction et complétion de phrases

Les modèles de bigrams et trigrams ont été utilisés pour prédire le mot suivant dans une phrase donnée, et pour compléter des phrases entières. Cela démontre l'efficacité des n-grams pour la génération de texte basée sur des séquences de mots précédents.

## 6.6 Conclusion

Le modèle N-gram, bien que simple, offre une capacité intéressante pour prédire et générer du texte basé sur les séquences de mots précédents. Cependant, il est limité par la taille du contexte qu'il peut utiliser (les n-grams). Des modèles plus avancés, comme les modèles de langage basés sur les réseaux de neurones, peuvent utiliser un contexte plus large et capturer des relations plus complexes dans les données textuelles.

## 7. Régression Logistique

### 7.1 TF-IDF

La *featurisation TF-IDF* est une méthode qui transforme les mots en vecteurs numériques, attribuant un poids à chaque mot selon sa fréquence dans le document et dans l'ensemble du corpus. Cette technique ne prend pas en compte la séquence des mots ni les relations entre eux. Malgré cette limitation, elle se révèle très efficace lorsqu'elle est utilisée avec des modèles de régression linéaire.

#### Premiers Résultats

Étant donné le déséquilibre du jeu de données, le paramètre `class_weight` a été ajusté à `balanced` pour compenser cette disparité. Des tests ont été réalisés sur les douze variantes de prétraitement du jeu de données. Voici les résultats obtenus pour le modèle de régression logistique sans optimisation des hyperparamètres :

TABLE 7.1 – Score Macro de la Régression Logistique TF-IDF, sur le jeu de validation. (3 meilleurs)

Technique de Prétraitement	Précision	Rappel	Score F1	Support
BPE Tokenizer 2	<b>0.826</b>	<b>0.911</b>	<b>0.862</b>	31915
BPE Tokenizer 1	0.822	0.910	0.859	31915
GPT Tokenizer 1	0.820	0.910	0.857	31915
Baseline	0.790	0.887	0.8318	31915

#### Optimisation des hyperparamètres

Afin d'optimiser les performances du modèle, une optimisation des hyperparamètres a été réalisée sur la meilleure méthode de prétraitement *BPE Tokenizer 2*. Les meilleurs paramètres obtenus sont une régularisation de type *L1* avec un paramètre de régularisation *C* égal à *0.60*.

TABLE 7.2 – Score Macro de la Régression Logistique TF-IDF Optimisé, sur le jeu de validation. (3 meilleurs)

Technique de Prétraitement	Précision	Rappel	Score F1	Support
BPE Tokenizer 2	<b>0.826</b>	<b>0.916</b>	<b>0.863</b>	31915
GPT Tokenizer 1	0.820	0.915	0.859	31915
BPE Tokenizer 1	0.821	0.914	0.860	31915
Baseline	0.784	0.890	0.825	31915

## Résultat sur le jeu de test

TABLE 7.3 – Score Macro de la Régression Logistique TF-IDF, sur le jeu de test

Technique de Prétraitement	Précision	Rappel	Score F1	Support
BPE Tokenizer 2	<b>0.706</b>	<b>0.900</b>	<b>0.751</b>	63978
Baseline	0.695	0.869	0.737	63978

## Interprétation des résultats

### Faux positifs

L'examen des faux positifs révèle la présence de termes offensants, confirmant ainsi une labellisation imparfaite. Un nuage de mots des faux positifs est disponible en annexe C.

### Faux négatifs

Il est à noter que certains termes négatifs sont plus fréquents dans le jeu de données de test que dans celui d'entraînement, ce qui contribue aux erreurs du modèle. Par exemple, le terme “boob” apparaît 1000 fois dans le jeu de données de test contre 32 fois dans celui d'entraînement. Un nuage de mots des faux négatifs est également disponible en annexe C.

### SHAP Values

Les *SHAP Values* révèlent les mots ayant le plus d'impact sur le modèle. Sans surprise, les termes les plus influents sont les plus vulgaires. Les mots comme “please” ou “thank” ont un impact négatif sur la prédiction de toxicité. Un aperçu des 20 mots avec les plus grandes *SHAP Values* est présenté en annexe C.

## 7.2 Word2Vec

La *featurisation Word2Vec* est une technique qui permet de convertir les mots en vecteurs de nombres. Elle permet de réduire la dimensionnalité des données et de capturer les relations sémantiques entre les mots.

### Comparaison Pré-Entraîné et Entraîné de zéro

Une comparaison a été effectuée entre la représentation *Word2Vec* d'un modèle pré-entraîné et celle d'un modèle entraîné à partir de notre jeu de données. Le modèle entraîné à partir de zéro repose sur **200** dimensions pour être comparable au modèle pré-entraîné. Il en ressort que la régression logistique est nettement meilleure avec un modèle pré-entraîné. Les moins bons résultats du modèle entraîné à partir de zéro peuvent s'expliquer par un manque de données pour exprimer la sémantique des mots.

TABLE 7.4 – Score Macro de la Régression Logistique Word2Vec, sur le jeu de validation

Technique de Prétraitement	Précision	Rappel	Score F1	Support
<b>Pré-Entraîné</b>				
Word Tokenizer 1	<b>0.744</b>	<b>0.884</b>	<b>0.790</b>	31915
BPE Tokenizer 2	0.740	0.880	0.784	31915
Baseline	0.637	0.810	0.654	31915
<b>Entraîné</b>				
Word Tokenizer 3	<b>0.677</b>	<b>0.863</b>	<b>0.712</b>	31915
BPE Tokenizer 1	0.674	0.862	0.708	31915
Baseline	0.650	0.809	0.676	31915

Ce tableau comparatif soulève plusieurs points. L'importance des prétraitements des données est plus marquée que pour la *featurisation TF-IDF*. De plus, les prétraitements *GPT Tokenizer* sont moins performants avec le modèle *Word2Vec* pré-entraîné que les prétraitements *word Tokenizer*. Cela s'explique par le fait que le modèle *Word2Vec* pré-entraîné attend des mots et non des *byte tokens*.

### Optimisation des hyperparamètres

La régression logistique a été optimisée en utilisant la meilleure technique de prétraitement *Word Tokenizer 1*, vectorisée par le modèle *Word2Vec* pré-entraîné. Lors de l'optimisation, il n'a pas été possible de tester la régularisation *L1* car le modèle prenait trop de temps à converger. Le meilleur ensemble de paramètres inclut une régularisation *L2* avec un *C* égale à 27.26.



## Résultat sur le jeu de test

TABLE 7.5 – Score de la Régression Logistique Word2Vec Optimisé, sur le jeu de test

Technique de Prétraitement	Précision	Rappel	Score F1	Support
Word Tokenizer 3	0.673	<b>0.870</b>	0.706	63978
Word Tokenizer 1	<b>0.680</b>	0.859	<b>0.716</b>	63978
Baseline	0.637	0.810	0.655	63978

Il est notable que les résultats sont inférieurs à ceux obtenus avec une *featurisation TF-IDF*.

## Interprétation des résultats

### Faux positifs

Les faux positifs remontés apportent les mêmes conclusions qu’avec *TF-IDF*.

### Faux négatifs

Les faux négatifs sont cependant plus nombreux qu’avec la *featurisation TF-IDF*. Ils présentent également des différences. Certains mots longs, qui sont des insultes, sont mal prédits. Par exemple, le modèle n’a pas identifié ‘motherfucker’ comme toxique. Cela révèle deux problèmes : le *Tokenizer* ne sépare pas les mots composés et le modèle *Word2Vec* n’a pas été entraîné sur ces mots plus longs. Un ensemble de données plus grand et un entraînement de *Word2Vec* avec un *Tokenizer* entraîné sur le jeu d’entraînement auraient été nécessaires pour une meilleure compréhension globale de la sémantique.

### 7.3 Régression Multilabel

En raison de la nette différence de performance entre le modèle *Word2Vec* et *TF-IDF*, la *featurisation TF-IDF* a été choisie pour le modèle *multilabel*. Le prétraitement choisi est *BPE Tokenizer 2*, car il a obtenu les meilleurs scores en termes F1-score et rappel macro. Un modèle par label est entraîné.

#### Optimisation des hyperparamètres

Après optimisation des hyperparamètres pour chaque label voici les résultats :

TABLE 7.6 – Scores de Précision, Rappel, et F1-Score par Label, d’un modèle de Régression Logistique TF-IDF Optimisé pour chaque label, sur le jeu de test

Label	Précision	Rappel	F1-Score	Support
toxic	0.42	0.93	0.58	6090
severe toxic	0.10	0.92	0.19	367
obscene	0.43	0.91	0.59	3691
threat	0.09	0.90	0.16	211
insult	0.37	0.90	0.53	3427
identity hate	0.17	0.91	0.28	712
overall non toxic	0.99	0.84	0.91	57735
macro avg	0.37	0.90	0.46	72233
weighted avg	0.87	0.86	0.84	72233

Pour les label *threat*, *severe toxic* et *identity hate*, le modèle a un score de rappel nettement meilleur que la précision. Cela est dû au fait qu’il y a très peu d’échantillons correspondant à ce label dans le jeu de données. De plus, le modèle de régression logistique utilisant *TF-IDF* ne prend pas en compte la sémantique des mots. En conclusion, le modèle de régression logistique sera efficace pour identifier les éléments toxiques (rappel macro à **0,90**), mais moins efficace pour déterminer précisément le type de toxicité présent (précision macro à **0.37**).

## 8. Feed Forward Neural Networks

### 8.1 Étape de la construction du modèle

Pour construire un modèle de réseau de neurones, il est nécessaire de définir l'architecture du réseau, les entrées, les sorties et les fonctions d'activation, ainsi que les fonctions de coût. Pour ce faire, L'étude préalable a servi de base pour sélectionner les paramètres de manière optimale.

### 8.2 Les embeddings

Les *embeddings* sont une technique couramment utilisée pour représenter des données de manière dense et continue dans des espaces de grande dimension. Ils sont particulièrement utiles pour traiter des données textuelles et des catégories discrètes. Pour réaliser une classification multilabel, Il a été décidé d'utiliser deux types d'*embeddings* différents : *GloVe* et *FastText*.

#### 8.2.1 GloVe

*GloVe* (Global Vectors for Word Representation) est un algorithme de vectorisation de mots qui représente les mots sous forme de vecteurs de dimension prédéfinie. Il utilise des statistiques globales de cooccurrence des mots pour capturer des relations contextuelles subtiles, ce qui le rend particulièrement adapté à des tâches comme la classification de texte.

#### 8.2.2 FastText

*FastText* est un modèle d'*embeddings* développé par *Facebook AI Research (FAIR)* qui améliore les techniques de vectorisation de mots en prenant en compte les sous-mots. Contrairement aux méthodes traditionnelles qui représentent les mots entiers comme des unités indépendantes, *FastText* décompose chaque mot en n-grammes de caractères, ce qui permet de capturer les relations morphologiques entre les mots.

### 8.3 Architecture du modèle

Le modèle de réseau de neurones *feedforward (FNN)* construit est composé de plusieurs couches entièrement connectées. Voici les détails de l'architecture du modèle :

- **Couche d'entrée** : Reçoit les données d'entrée de dimension **200**.
- **Couche cachée** : Une couche entièrement connectée de dimension *hidden\_dim* suivie d'une normalisation par lots (**BatchNorm1d**).
- **Fonction d'activation** : *ReLU (Rectified Linear Unit)* après la couche cachée.
- **Couche de sortie** : Une couche entièrement connectée de dimension **6** suivie d'une fonction d'activation sigmoïde pour la classification multilabel de chaque catégorie.

### 8.3.1 Fonction de coût

Pour entraîner notre modèle, La fonction de coût utilisée est la vraisemblance négative logarithmique (*BCEWithLogitsLoss*) qui est couramment utilisée pour les tâches de classification multilabel. De plus, la pondération des classes a été utilisée pour compenser le déséquilibre des classes dans le jeu de données.

### 8.3.2 Optimisation

Pour optimiser le modèle, l'algorithme d'optimisation employé est *Adam* (*AdamOptimizer*), qui est une méthode d'optimisation stochastique basée sur l'estimation adaptative des moments.

## 8.4 Création des modèles

Afin de trouver la meilleure architecture pour notre modèle, il a été décidé de tester plusieurs configurations de réseaux de neurones en faisant varier trois paramètres :

- **Nombre de neurones dans la couche cachée** : **32,64,128**
- **Type d'embedding utilisé** : *GloVe* et *FastText*
- **Type de prétraitement des données textuelles** : Différentes méthodes de prétraitement des données textuelles ont été explorées, telles que la normalisation, la suppression des stop-words et la lemmatisation, afin de trouver la combinaison optimale pour notre tâche de classification multilabel.

## 8.5 Résultats

Voici les scores de précision, de rappel et de F1-Score par label, d'un modèle avec une couche cachée de 32 neurones, utilisant les *embeddings* de *FastText* et aucun prétraitement des données textuelles.

Après avoir entraîné et évalué plusieurs modèles de réseaux de neurones, il a été constaté que le modèle avec une couche cachée de 64 neurones, utilisant les *embeddings* de *GloVe* et un prétraitement des données textuelles comprenant la suppression des caractères spéciaux, la mise en minuscules et la suppression des duplications, a obtenu les meilleurs résultats.

TABLE 8.1 – Scores de Précision, Rappel, et F1-Score par Label, d’un modèle de réseau de neurones feedforward *FastText* Optimisé pour chaque label, sur le jeu de test

Label	Précision	Rappel	F1-Score	Support
toxic	0.64	0.50	0.56	6090
severe_toxic	0.27	0.22	0.24	367
obscene	0.71	0.46	0.56	3691
threat	0.50	0.07	0.12	211
insult	0.66	0.36	0.47	3427
identity_hate	0.50	0.02	0.04	712
overall_non_toxic	0.95	0.97	0.96	57735
macro avg	0.60	0.37	0.42	72233
weighted avg	0.89	0.86	0.87	72233

TABLE 8.2 – Scores de Précision, Rappel, et F1-Score par Label, d’un modèle de réseau de neurones feedforward *GloVe* Optimisé pour chaque label, sur le jeu de test

Label	Précision	Rappel	F1-Score	Support
toxic	0.55	0.72	0.62	6090
severe_toxic	0.33	0.47	0.38	367
obscene	0.62	0.63	0.62	3691
threat	0.52	0.42	0.46	211
insult	0.59	0.61	0.60	3427
identity_hate	0.49	0.37	0.42	712
overall_non_toxic	0.97	0.94	0.95	57735
macro avg	0.58	0.59	0.58	72233
weighted avg	0.89	0.88	0.88	72233

## 9. Transformers

### 9.1 Construction du modèle

Afin d'améliorer les performances de notre solution, nous avons décidé de tester un modèle *transformer*. Pour ce faire nous avons utilisé la bibliothèque *Hugging Face* qui propose des implémentations de modèles *transformer* préentraînés. Nous avons choisi d'utiliser le modèle *BERT* qui a été préentraîné sur des données textuelles massives et qui a montré de très bonnes performances sur de nombreuses tâches de traitement du langage naturel. Nous avons entraîné le modèle sur notre jeu de données pour réaliser une classification multilabel des commentaires à l'aide de *Google Colab*.

#### 9.1.1 Prétraitement des données

Pour prétraiter les données textuelles, nous avons utilisé le *tokenizer* *BERTTokenizer* qui permet de *transformer* les textes en *tokens* qui peuvent être compris par le modèle *BERT*.

#### 9.1.2 Architecture du modèle

Nous avons utilisé le modèle transformeur *BERT*, un modèle allégé de *BERT*, pour notre tâche de classification multilabel. Voici les spécificités de l'architecture du modèle :

- Il est constitué de plusieurs couches de transformeurs pour capturer les représentations contextuelles des mots.
- Le modèle a été adapté pour notre tâche avec une couche de sortie de dimension **6** et une fonction d'activation sigmoïde, permettant la classification multilabel des catégories.
- La fonction de coût utilisée est la perte de log-vraisemblance négative (*BCEWithLogitsLoss*), adaptée aux tâches de classification multilabel.
- L'optimisation a été réalisée à l'aide de l'algorithme Adam, une méthode efficace pour les grands ensembles de données et les réseaux de neurones complexes.

### 9.2 Résultats

Voici les scores de précision, de rappel et de F1-Score par label, d'un modèle *BERT* pré-entraîné sur des données textuelles massives, pour la classification multilabel des différentes catégories au bout de **10 epochs**.

TABLE 9.1 – Scores de Précision, Rappel, et F1-Score par Label, d’un modèle *BERT* **sans** la pondération des classes, sur le jeu de test

Label	Précision	Rappel	F1-Score	Support
toxic	0.52	0.91	0.66	6090
severe_toxic	0.40	0.38	0.39	367
obscene	0.59	0.82	0.69	3691
threat	0.46	0.57	0.51	211
insult	0.62	0.78	0.69	3427
identity_hate	0.63	0.61	0.62	712
overall_non_toxic	0.99	0.91	0.95	57735
macro avg	0.60	0.71	0.64	72233
weighted avg	0.90	0.90	0.89	72233

TABLE 9.2 – Scores de Précision, Rappel, et F1-Score par Label, d’un modèle *BERT* **avec** la pondération des classes, sur le jeu de test

Label	Précision	Rappel	F1-Score	Support
toxic	0.52	0.91	0.66	6090
severe_toxic	0.40	0.38	0.39	367
obscene	0.59	0.82	0.69	3691
threat	0.46	0.57	0.51	211
insult	0.62	0.78	0.69	3427
identity_hate	0.63	0.61	0.62	712
overall_non_toxic	0.99	0.91	0.95	57735
macro avg	0.60	0.71	0.64	72233
weighted avg	0.90	0.90	0.89	72233

## 10. Limitation et Perspectives

### 10.1 Limitations

#### Machine Learning

Le jeu de données de test n'étant pas labellisé correctement, les modèles de *machine learning* avaient un peu plus de mal à généraliser et prédire correctement les *labels* des commentaires. De plus, le jeu de données étant déséquilibré, les modèles avaient tendance à mieux prédire les classes majoritaires.

#### Deep Learning

Concernant les modèles *transformers*, une limitation majeure est le temps de calcul nécessaire pour entraîner ces modèles sur des jeux de données massifs. En effet, l'entraînement de ces modèles peut prendre plusieurs heures, voire plusieurs jours, en fonction de la taille du jeu de données et de la complexité du modèle. C'est pour cela que le service *Google Colab Pro* et le *GPU A100* ont été utilisés. Celui-ci étant payant, la quantité de test était restreinte.

### 10.2 Perspectives

#### Data augmentation

Faire une augmentation de données pour équilibrer le jeu de données aurait permis d'améliorer les performances des modèles. Ainsi, il aurait été possible de traduire les phrases en français, allemand ou espagnol pour augmenter la taille du jeu de données. Un *Large Language Model (LLM)* aurait pu être utile pour créer des données synthétiques afin d'améliorer les performances des modèles.

Le jeu de données fourni aurait aussi pu être augmenté par un autre jeu de données afin d'avoir d'autres références toxiques et un modèle plus général.

#### Fine-tuning

Pour les modèles de *transformers*, il aurait été intéressant d'exploiter d'autres modèles d'*encoder* pré-entraînés comme *Roberta*, *DeBERTa*, *DistilBERT* ou *CANINE*, ainsi que de comparer différents hyperparamètres.



## A. Appendix

### Tableau représentatif des différentes fonctions de prétraitement

Pré traitement	Remove special caractère	Remove Stop Words	Replace emojis	Lowercase	Lemmatization	Remove punctuation	Remove duplication
BPE Tokenizer 2	VRAI	FAUX	FAUX	VRAI	FAUX	VRAI	VRAI
BPE Tokenizer 1	VRAI	FAUX	FAUX	VRAI	FAUX	FAUX	VRAI
Word Tokenizer 4	VRAI	VRAI	VRAI	VRAI	VRAI	VRAI	VRAI
Word Tokenizer 3	VRAI	VRAI	VRAI	VRAI	VRAI	FAUX	FAUX
Word Tokenizer 2	VRAI	FAUX	FAUX	VRAI	FAUX	VRAI	FAUX
Word Tokenizer 1	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
GPT Tokenizer 4	VRAI	VRAI	VRAI	VRAI	VRAI	VRAI	VRAI
GPT Tokenizer 3	VRAI	VRAI	VRAI	VRAI	VRAI	FAUX	FAUX
GPT Tokenizer 2	VRAI	FAUX	FAUX	VRAI	FAUX	FAUX	FAUX
GPT Tokenizer 1	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
Baseline	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX

TABLE A.1 – Pré traitement comparatif

## B. Appendix

### B.1 Corrélation entre les labels

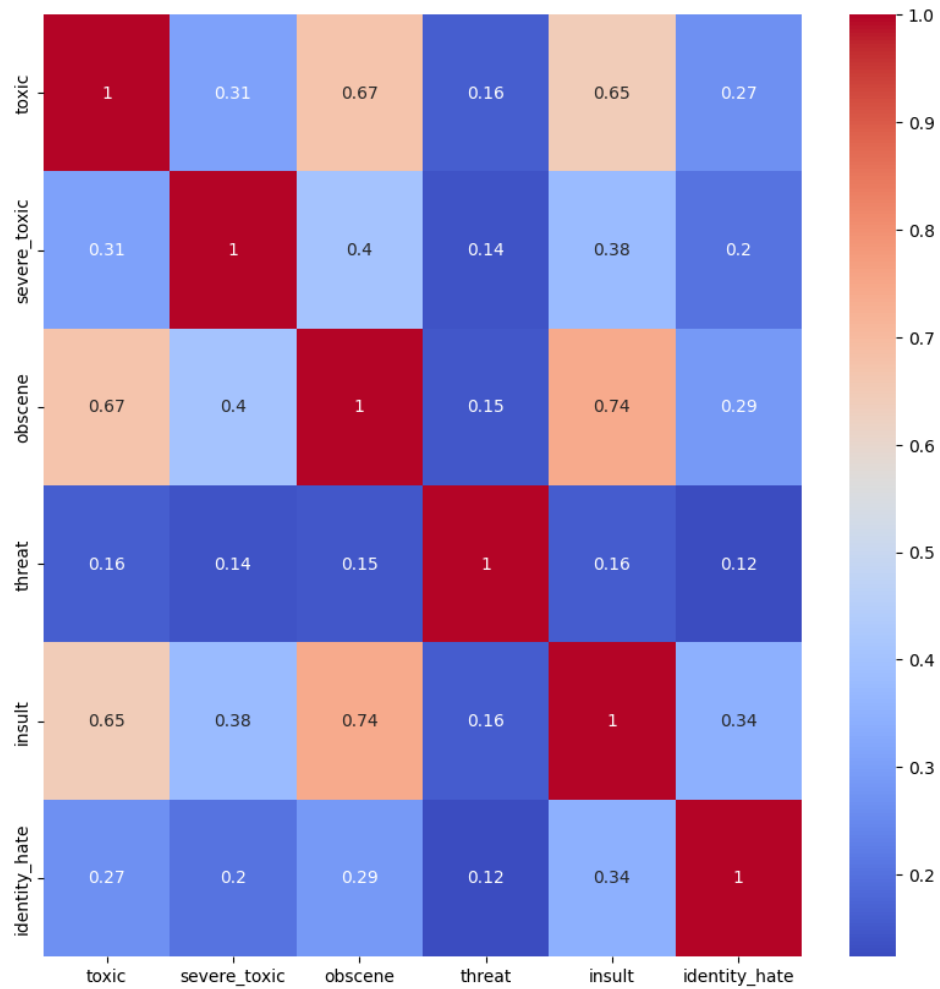


FIGURE B.1 – Matrice de corrélation des labels du dataset







### C.3 Shap Values

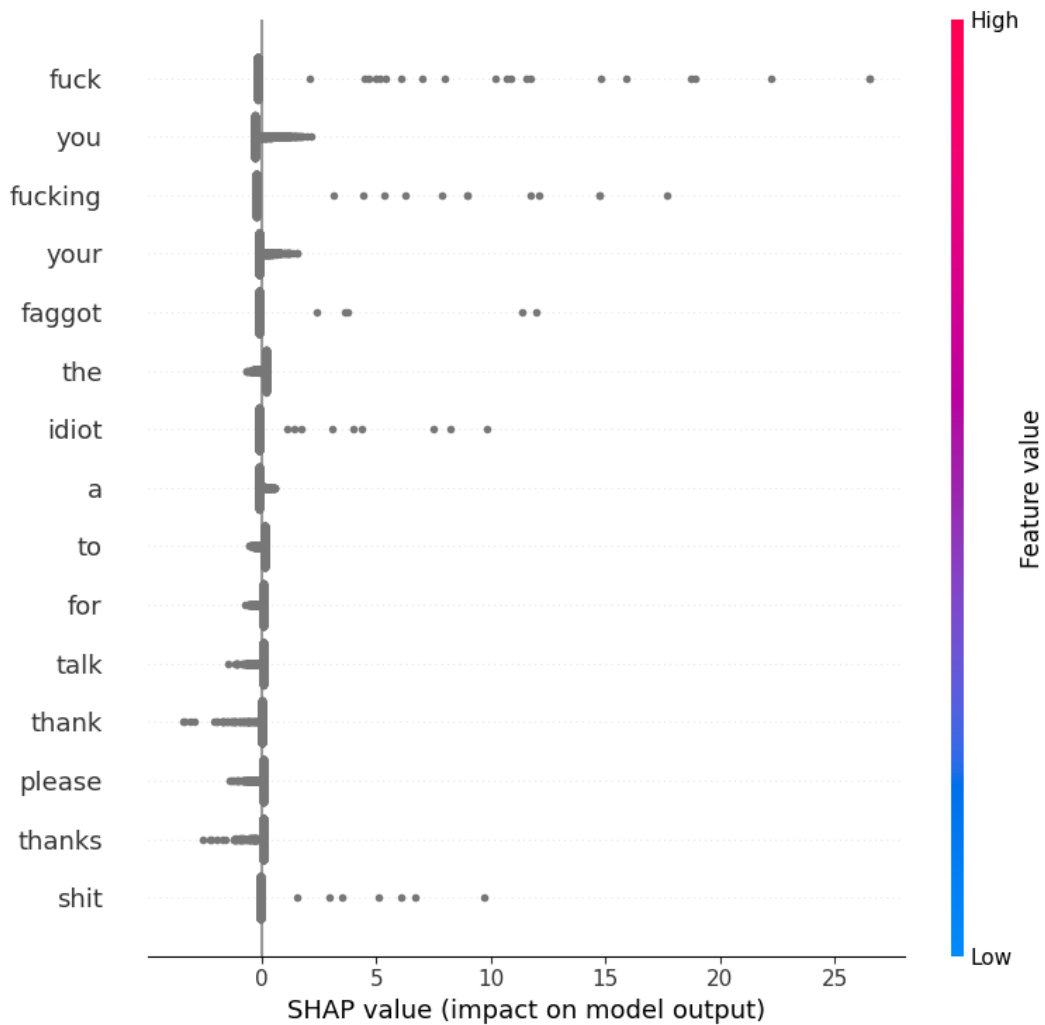


FIGURE C.5 – Shap Values – Régression Logistique TF-IDF Optimisé, sur le jeu de test.  
Pré-traitement : BPE Tokenizer 2

## Bibliographie

[van Aken et al.(2018)van Aken, Risch, Krestel, and Loser] Betty van Aken, Julian Risch, Ralf Krestel, and Alexander Loser. Challenges for toxic comment classification : An in-depth error analysis. *arXiv preprint arXiv :1809.07572*, 2018. URL <https://arxiv.org/pdf/1809.07572>.