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'intermediaire de modèle NLP

Table des matières

1 Introduction **1**

2 Présentation du jeu de données **2**

3 Statistiques descriptives du jeu de données **4**

4 Tokenizer **6**

4.1 Prétraitement du jeu de données 6

4.1.1 Tokenisation à base d'expressions régulières (RegexTokenizer) 6

4.1.2 Tokenisation byte-pair encoding (*TikToken*) 6

4.1.3 Comparaison avec d'autres tokenizers 7

4.1.4 Méthodes de normalisation du texte 7

4.1.5 Prétraitement supplémentaire du jeu de données 9

5 Naive Bayes **12**

5.1 Pré-traitement des données 12

5.1.1 Tokenisation des commentaires 12

5.2 Analyse exploratoire des données 12

5.2.1 Statistiques descriptives 12

5.3 Entraînement du modèle 12

5.3.1 Préparation des données 12

5.3.2 Évaluation des performances du modèle 12

5.3.3 Validation croisée 13

5.4 Analyse des résultats 13

5.4.1 Calcul des probabilités 13

5.4.2 Calcul des vraisemblances 13

5.4.3 Prédiction 13

5.4.4 Matrice de confusion 13

5.4.5 Etudes Annexe sur les données 14

6 N-gram **16**

6.1 Pré-traitement des données 16

6.1.1 Tokenisation des commentaires 16

6.2 Analyse exploratoire des données 16

6.2.1 Statistiques descriptives 16

6.3 Entraînement du modèle 16

6.3.1 Préparation des données 16

6.3.2 Création du modèle de bigrams et trigrams 17

6.3.3 Fréquence des n-grams 17

6.3.4 Fonction de prédiction 17
6.4 Évaluation des performances du modèle 18
6.5 Analyse des résultats 18
6.5.1 Interprétation des fréquences des n-grams 18
6.5.2 Prédiction et complétion de phrases 18
6.6 Conclusion 18

7 Régression Logistique **19**
7.1 TF-IDF 19
7.2 Word2Vec 21
7.3 Régression Multilabel 22

8 Feed Forward Neural Networks **25**
8.1 Étape de la construction du modèle 25
8.2 Les Embeddings 25
8.2.1 GloVe 25
8.2.2 FastText 25
8.3 Architecture du modèle 25
8.3.1 Fonction de coût 26
8.3.2 Optimisation 26
8.4 Création des modèles 26
8.5 Résultats 26

9 Transformers **28**
9.1 Construction du modèle 28
9.1.1 Prétraitement des données 28
9.1.2 Architecture du modèle 28
9.2 Résultats 28

A Appendix **30**
A.1 Corrélation entre les labels 30
A.2 Distribution des mots 31

B Appendix **32**
B.1 Word Cloud Faux Positifs 32
B.2 Word Cloud Faux Négatifs 33
B.3 Shap Values 34

1. Introduction

Ce projet vise à développer des modèles de traitement du language 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 : jigswaw 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 dataset 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 dataset 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 dataset 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 dataset 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 dataset peuvent fonctionner de manière efficace et équitable.

Distribution

Le dataset 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 dataset 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. Il y a aussi une répartition inégale au sein des labels de toxicité La somme des pourcentages n'est pas égale à 100% car un commentaire peut avoir plusieurs labels. C'est donc dans 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

On peut remarqué que les labels de toxicité sont fortement corrélés entre eux. On peut dès à présent anticiper une difficulté du modèle à distinge un commentaire toxique d'un commentaire obscene (**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 A.

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 problèmes lors de la conception du modèle. Il est donc important de prétraiter les données pour normaliser la longueur des commentaires.

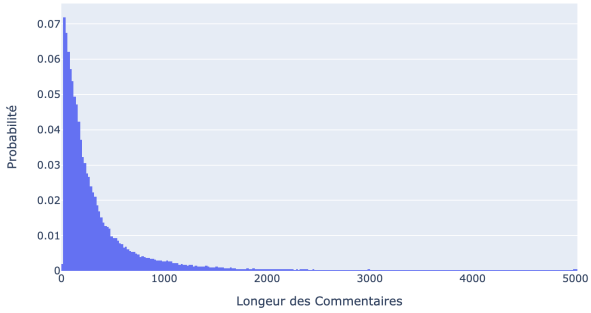

FIGURE 3.1 – Distribution de la longueur des commentaires

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 dataset, 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 A, 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, nous appliquerons différentes techniques de prétraitement sur notre jeu de données afin de le rendre apte à être utilisé dans nos modèles prédictifs.

4.1.1 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.

Implémentation de la tokenisation à l'aide de la classe RegexTokenizer

Nous commençons par importer la classe RegexTokenizer et appliquer la tokenisation sur notre 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

4.1.2 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.

Implémentation de la tokenisation à l'aide de la bibliothèque *TikToken*

Nous appliquons la tokenisation BPE sur notre jeu de données en utilisant la bibliothèque *TikToken*.

Avant tokenisation	**Après tokenisation**
"Hello, World !"	['Hello', ',', 'World', ' !']
"I love *TikToken*"	['I', 'love', 'Ti', 'k', 'To', 'ken']

TABLE 4.2 – Exemple de tokenisation BPE

4.1.3 Comparaison avec d'autres tokenizers

Nous avons également comparé la performance de notre tokenizer avec d'autres options disponibles dans la bibliothèque MinBPE.

Tokenizer	**Temps d'entraînement (minutes)**
RegexTokenizer	182
TikToken	0 (déjà entrainé)
BasicTokenizer	69

TABLE 4.3 – Comparaison des temps d'entraînement des tokenizers

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

4.1.4 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', ...). Nous les supprimerons de notre 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

Nous avons modifié la méthode de lemmatisation 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, nous avons 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, nous avons décidé 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 nous avons pris la décision de remplacer certains emojis par un mot le représentant au mieux afin de permettre aux models de mieux s'adapter.

Effacement des ponctuations

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

Effacement des duplications

En constatant que plusieurs messages vulgaires étaient dissimulés parmi nos tokens en raison de la duplication des premières et/ou dernières lettres, nous avons été contraints de corriger ces tokens pour leur attribuer une plus grande valeur.

Tableau représentatif des différentes fonctions de prétraitement

Pré traitement	Remove special caractère	Remove Stop W	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 4.4 – Pré traitement comparatif

4.1.5 Prétraitement supplémentaire du jeu de données

En plus des techniques de prétraitement déjà mentionnées, nous avons également effectué les actions suivantes pour rendre notre jeu de données plus adapté à l'entraînement de nos modèles :

Réflexion sur la suppression des symboles opérateurs et des chiffres

Dans le cadre du prétraitement des données, nous avons envisagé initialement de supprimer les symboles opérateurs et les chiffres présents dans nos textes. Cependant, après réflexion, nous avons 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, nous avons mis en place un processus sophistiqué consistant à utiliser une série d'expressions régulières (regex) pour remplacer ces symboles par leur signification correspondante. Par exemple, nous avons remplacé des symboles comme " :/" par leur signification afin d'améliorer la compréhension du texte par nos modèles. Cette approche nous 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	laught
8==========>	penis

TABLE 4.5 – Exemple d'éléments toxiques contenant des symboles opérateurs et des chiffres que nous avons remplacé par leur signification

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

Nous avons supprimé les balises HTML, les URL et les adresses e-mail de notre 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

Nous avons utilisé un algorithme de correction des erreurs orthographiques et des fautes de frappe pour améliorer la qualité de nos 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 'ı' 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**
lıes	lies
penisd i'\m	penis i'm
fùck8	fuck

TABLE 4.6 – Exemple de phrase avant et après correction

Non-application de la correction d'orthographe

Initialement, nous avions envisagé d'appliquer un algorithme de correction d'orthographe pour améliorer la qualité de nos données textuelles. Cependant, après avoir réalisé que cette étape prenait trop de temps et pouvait introduire des erreurs, notamment en modifiant des noms propres en mots n'ayant aucun rapport, nous avons décidé 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, nous avons choisi de ne pas effectuer cette étape de correction d'orthographe pour garantir l'intégrité et la qualité de nos données textuelles.

Exemple d'une phrase après toutes les étapes de normalisation

Étape	**Texte normalisé**
Texte initial	motherrrrrrrrrrrrFuckkkkkkkk!!!!aaaaaaa***** stringssss ***!!!!!!!bitchhhhhh :) 8=========
Après remplacement des emojis	motherrrrrrrrrrrrFuckkkkkkkk!!!!aaaaaaa***** stringssss ***!!!!!!!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 4.7 – 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 nôtre 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 "chretian 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.

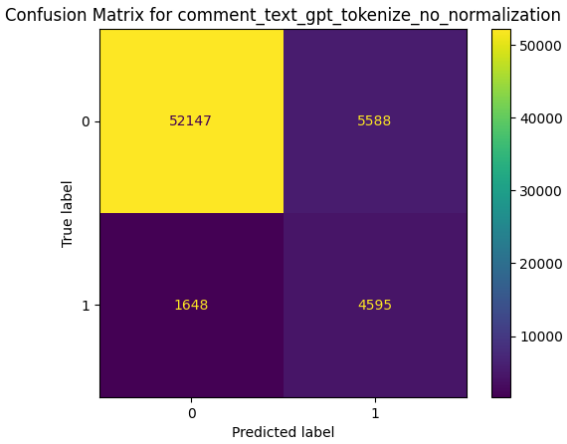

FIGURE 5.1 – Matrice de confusion du modèle Bayésien naïf

5.4.5 Etudes Annexe sur les données

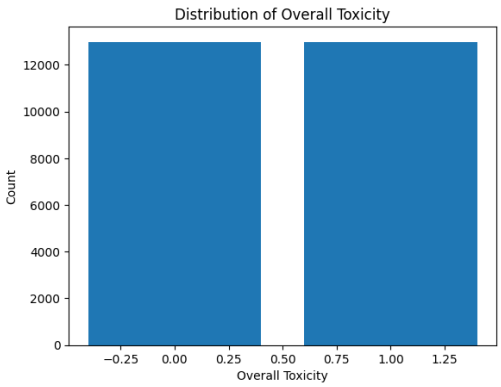

FIGURE 5.2 – Distribution de la toxicité des commentaires

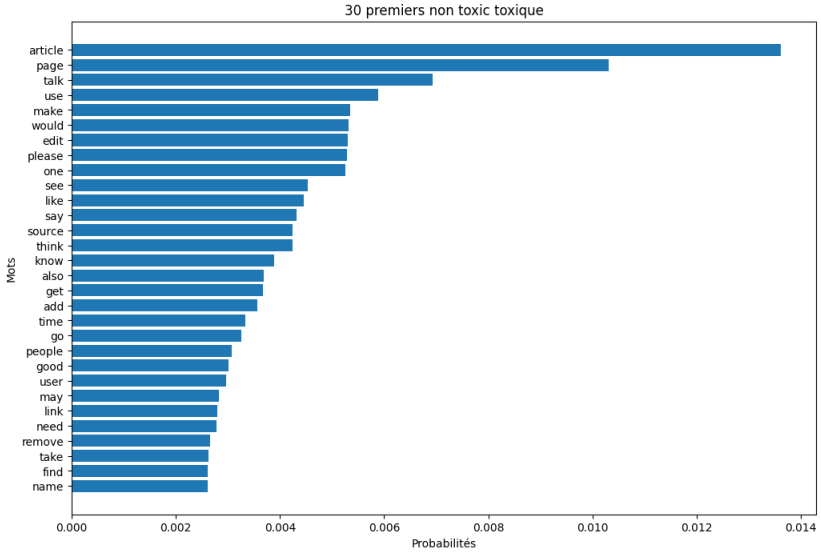

FIGURE 5.3 – Top 30 mots non toxiques

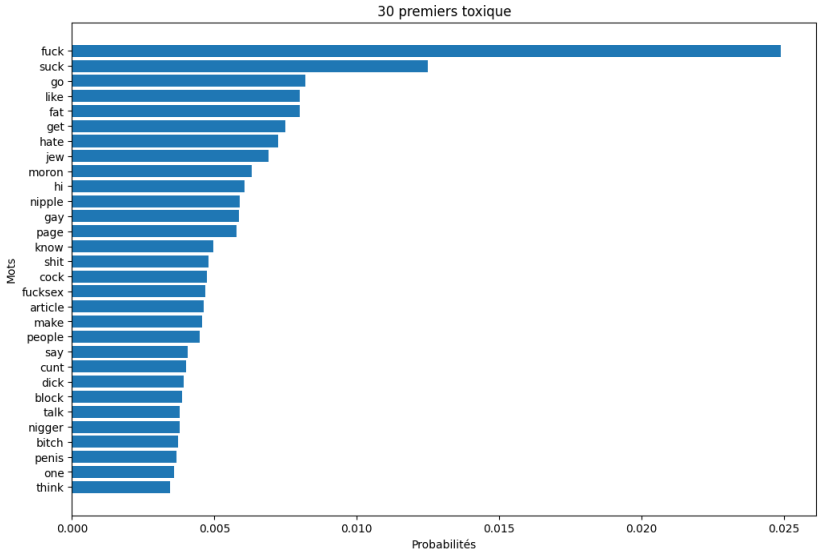

FIGURE 5.4 – Top 30 mots toxiques

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

Choix des Métrics

Avant de débuter l'analyse avec le modèle, il a été décidé de sélectionner des métriques d'évaluation telles que le rappel macro et le F1-score macro pour les labels *toxique*. En utilisant un modèle de régression linéaire, la compréhension des phrases peut être difficile et entraîner une précision réduite, surtout avec une labellisation imparfaite et des faux positifs. Cependant, il est crucial de maximiser le rappel pour détecter un maximum de phrases toxiques, quitte à accepter quelques erreurs. Le choix de la pondération macro a été privilégié pour une évaluation globale de la performance, en donnant le même poids à chaque classe. Ainsi, lors de chaque optimisation des paramètres, la métrique de rappel macro a été maximisée.

Étape de la construction du modèle

La classification *multilabel* a été simplifiée en classification binaire par la création d'un label unique nommé *overall toxic*, considéré comme vrai si au moins un des labels toxiques est présent. Deux méthodes de featurisation et douze méthodes de pré-traitement ont été explorées. Pour chaque featurisation, divers hyper-paramètres ont été optimisés à l'aide de la bibliothèque `Optuna`. Enfin, un modèle de régression linéaire *multilabel* a été entraîné sur les meilleures méthodes de pré-traitement et de featurisation.

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.

Premier 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 pré-traitement du jeu de données. Voici les résultats obtenus pour le modèle de régression logistique sans optimisation des hyperparamètres :

Optimisation des hyper-paramètres

Afin d'optimiser les performances du modèle, une optimisation des hyper-paramètres a été réalisé sur la meilleure méthode de pré-traitement *BPE Tokenizer 2*. Les meilleurs paramètres obtenues sont une régularisation de type *L1* avec un paramètre de régularisation C égal à *0.60*.

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

Technique de Pré-Traitement	Précision	Rappel	Score F1	Support
BPE Tokenizer 2	**0.826**	**0.911**	**0.862**	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

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

Technique de Pré-Traitement	Précision	Rappel	Score F1	Support
BPE Tokenizer 2	**0.826**	**0.916**	**0.863**	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 Pré-Traitement	Précision	Rappel	Score F1	Support
BPE Tokenizer 2	**0.706**	**0.900**	**0.751**	63978
Baseline	0.695	0.869	0.737	63978

La meilleur technique de pré-traitement reste le *BPE Tokenizer 2* avec ces paramètres de régularisation. On remarque néanmoins que les résultats sur le jeu de test sont moins bons que sur le jeu de validation.

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 B.

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 B.

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. On remarque aussi que 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 B.

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 Entrainé à partir de zéro

Une comparaison a été effectué entre la representation *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 regréssion logistique est nettement meilleur avec un modèle pré-entraîné. Les moins bons résultats du modèle entrainé 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 Pré-Traitement	**Précision**	**Rappel**	**Score F1**	**Support**
Pré-Entraîné				
Word Tokenizer 1	**0.744**	**0.884**	**0.790**	31915
BPE Tokenizer 2	0.740	0.880	0.784	31915
Baseline	0.637	0.810	0.654	31915
Entraîné				
Word Tokenizer 3	**0.677**	**0.863**	**0.712**	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 pré-traitements des données est plus marquée que pour la featurisation *TF-IDF*. De plus, les pré-traitement *GPT Tokenizerr* sont moins performants avec le modèle *word2vec* pré-entraîné que les pré-traitement

word Tokenizer. Cela s'explique par le fait que le modèle *word2vec* pré-entraîné attend des mots et non des *byte tokens*.

Résultat après optimisation des hyper-paramètres

La régression logistique a été optimisée en utilisant la meilleure technique de pré-traitement *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 Pré-Traitement	Précision	Rappel	Score F1	Support
Word Tokenizer 3	0.673	**0.870**	0.706	63978
Word Tokenizer 1	**0.680**	0.859	**0.716**	63978
Baseline (Sans Pré-traitement)	0.637	0.810	0.655	63978

On peut noter que les résultats sont moins bons qu'à partir d'une featurisation TF-IDF.

Interprétation des résultats

Faux positifs

On peut tirer les mêmes conclusions qu'avec *TF-IDF*.

Faux negatifs

Les faux négatifs sont cependant plus nombreux qu'avec la featurisation *TF-IDF*. Ils présentent également des différences. On remarque que 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 Tokenizerr 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 Tokenizerr 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 de macro et de F1-score rappel. Un modèle est entraîné par label.

Voici ici le score pour chaque label sans optimsation des hyper-paramètres.

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

Label	**Précision**	**Rappel**	**F1-Score**	**Support**
toxic	0.44	0.91	0.59	6090
severe_toxic	0.14	0.87	0.24	367
obscene	0.48	0.87	0.62	3691
threat	0.21	0.78	0.33	211
insult	0.40	0.87	0.55	3427
identity_hate	0.25	0.84	0.38	712
overall_non_toxic	0.99	0.87	0.93	57735
macro avg	0.41	0.86	0.52	72233
weighted avg	0.88	0.88	0.86	72233

Optimisation des hyper-paramètres

Apres optimisation des hyperparmaètre pour chaque label on obtient les résultats suivants :

TABLE 7.7 – 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

On remarque que pour le label 'threat', en améliorant le rappel, la précision a nettement diminué. Cela est dû au fait qu'il y a très peu de d'échantillon 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 conlusion, 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 et les sorties, les fonctions d'activation, ainsi que les fonctions de coût. Pour ce faire, nous nous sommes basés sur notre étude préalable sur la régression logistique afin de sélectionner au mieux nos paramètres.

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, nous avons 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 co-occurrence 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) que nous avons 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 batch (`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, nous avons utilisé la fonction de coût de la perte de log-vraisemblance négative (`BCEWithLogitsLoss`) qui est couramment utilisée pour les tâches de classification multilabel. De plus, nous avons utilisé la pondération des classes pour compenser le déséquilibre des classes dans notre jeu de données.

8.3.2 Optimisation

Pour optimiser notre modèle, nous avons utilisé l'algorithme d'optimisation 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, nous avons 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, nous avons 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 librairie Hugging Face qui propose des implémentations de modèles transformer pré-entraînés. Nous avons choisi d'utiliser le modèle DistilBert qui est une version plus légère du modèle BERT qui a été pré-entraîné sur des données textuelles massives. Cette version plus légère permet de réduire le temps d'entraînement et la consommation de mémoire tout en conservant 97% des performances du modèle BERT.

9.1.1 Prétraitement des données

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

9.1.2 Architecture du modèle

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

- DistilBERT conserve les performances de BERT tout en étant plus rapide et plus léger.
- 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 DistilBERT pré-entraîné sur des données textuelles massives, pour la classification multilabel des différentes catégories au bout de 10 époques.

TABLE 9.1 – Scores de Précision, Rappel, et F1-Score par Label, d'un modèle DistilBERT **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 DistilBERT **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

A. Appendix

A.1 Corrélation entre les labels

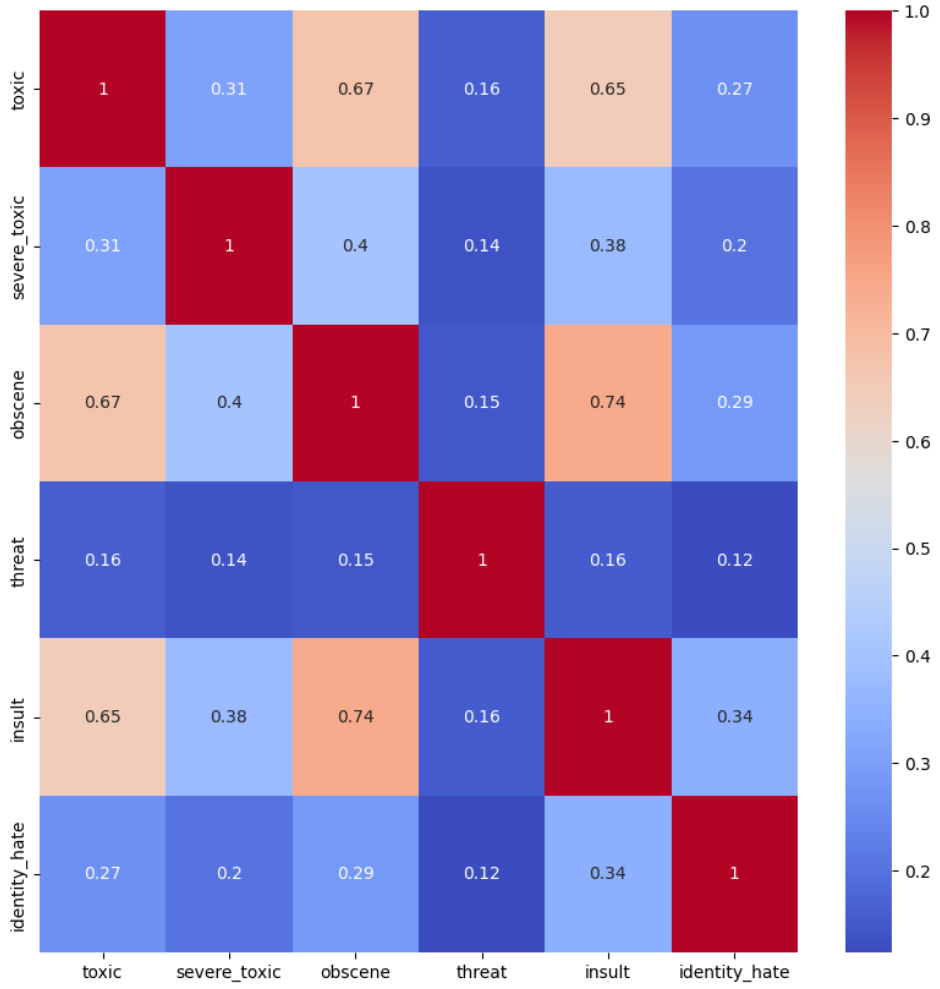

FIGURE A.1 – Matrice de corrélation des labels du dataset

A.2 Distribution des mots

(a) Word Cloud – Toxique

(b) Word Cloud – Haine identitaire

(c) Word Cloud – Obscene

(d) Word Cloud – Insult

FIGURE A.2 – Word Clouds

B. Appendix

B.1 Word Cloud Faux Positifs

FIGURE B.1 – Word Cloud Faux Positifs – Régression Logisitique TF-IDF Optimisé, sur le jeu de test. Pré-traitement : BPE Tokenizer 2

FIGURE B.2 – Word Cloud Faux Positifs – Régression Logisitique TF-IDF Optimisé, sur le jeu de test. Pré-traitement : Word Tokenizer 1

B.2 Word Cloud Faux Négatifs

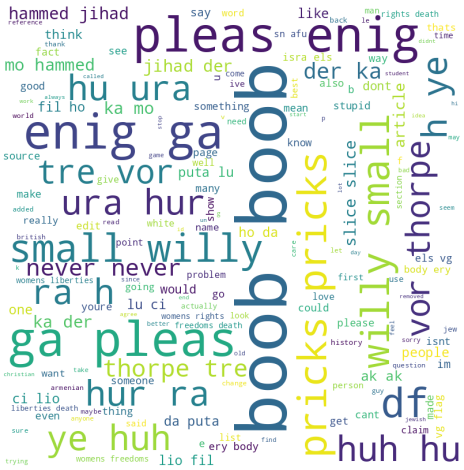

FIGURE B.3 – Word Cloud Faux Négatifs – Régression Logisitique TF-IDF Optimisé, sur le jeu de test. Pré-traitement : BPE Tokenizer 2

FIGURE B.4 – Word Cloud Faux Négatifs – Régression Logisitique Word2Vec Optimisé , sur le jeu de test. Pré-traitement : Word Tokenizer 1

B.3 Shap Values

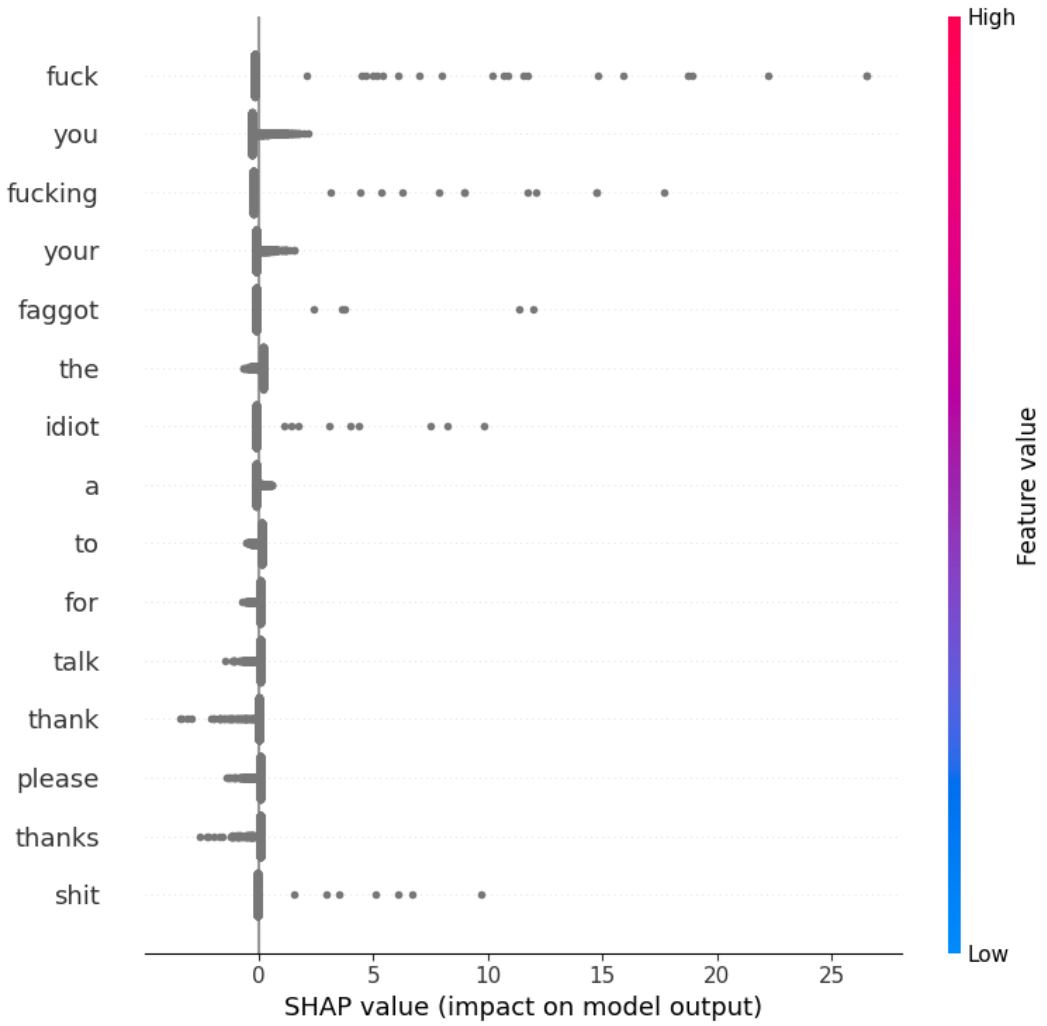

FIGURE B.5 – Shap Values – Régression Logisitique TF-IDF Optimisé, sur le jeu de test. Pré-traitement : BPE Tokenizer 2

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.