**Rapport sur la Classification de Commentaires Toxiques**

**Introduction**

Pour une entreprise proposant à ses utilisateurs d’écrire des commentaires, il est intéressant de savoir analyser les traces que laissent les utilisateurs sur leur plateforme. C’est d’autant plus important car vous pourriez être responsable autant que ceux qui l’ont écrit des propos injurieux et toxiques de vos utilisateurs. La classification de ces commentaires repose sur des techniques de traitement du langage naturel (NLP) et de machine learning. Dans ce contexte, nous avons mis en place une approche évolutive en trois étapes :

1. Un modèle de base utilisant la vectorisation TF-IDF associé à une régression logistique,
2. Une première approche de deep learning basée sur des réseaux de neurones récurrents (RNN) avec LSTM unidirectionnel,
3. Un modèle final optimisé intégrant une vectorisation intégrée et un LSTM bidirectionnel couplé à plusieurs couches denses.

Chaque étape permet d’identifier les forces et faiblesses de la méthode précédente et d’apporter des améliorations pour mieux capturer le sens des commentaires et traiter efficacement la classification multi-label.

**Partie 1 : Modèle 1 – Approche Basique avec TF-IDF**

**1.1 Présentation du Modèle**

Le Modèle 1 repose sur une approche classique en traitement du langage naturel, consistant à transformer les commentaires en représentations numériques à l'aide de la méthode TF-IDF (Term Frequency-Inverse Document Frequency).  
**Comment fonctionne TF-IDF ?**

* **Term Frequency (TF) :** Cette composante mesure la fréquence d’apparition d’un mot dans un document.
* **Inverse Document Frequency (IDF) :** Cette composante évalue l’importance d’un mot dans l’ensemble du corpus en réduisant le poids des termes fréquents et en amplifiant celui des termes rares.  
  En combinant ces deux mesures, TF-IDF attribue à chaque mot un score indiquant son pouvoir discriminant. L’idée est que les mots qui apparaissent fréquemment dans un document mais rarement dans le corpus global sont susceptibles d’être particulièrement pertinents pour la classification.

Dans ce contexte, les commentaires sont d’abord nettoyés (mise en minuscules et suppression des caractères spéciaux) pour uniformiser les données. Ensuite, la fonction **TfidfVectorizer** est utilisée avec une limite fixée à 10 000 caractéristiques pour convertir chaque commentaire en un vecteur numérique. Chaque dimension du vecteur représente le score TF-IDF d’un mot donné.

Après la vectorisation, le problème est traité comme une tâche de classification multi-label (chaque commentaire peut appartenir simultanément à plusieurs catégories comme *toxic*, *severe\_toxic*, *obscene*, *threat*, *insult*, et *identity\_hate*). Pour ce faire, on utilise un classificateur OneVsRest basé sur la régression logistique. Dans ce schéma, un modèle de régression logistique est entraîné pour chaque classe afin de prédire si un commentaire appartient ou non à celle-ci.

**1.2 Points Forts et Limitations**

**Avantages :**

* **Simplicité et rapidité :**
  + La mise en place de cette approche est relativement simple et rapide.
  + L’entraînement d’un modèle de régression logistique sur des vecteurs TF-IDF est peu coûteux en temps de calcul, ce qui permet des itérations rapides pour un premier benchmark.
* **Interprétabilité :**
  + La méthode TF-IDF offre une certaine transparence, puisque les poids attribués aux termes peuvent être interprétés.
  + Cela permet d’identifier quels mots influencent le plus la prédiction, ce qui est utile pour comprendre et expliquer le comportement du modèle.

**Limitations :**

* **Approche « Sac de Mots » (Bag-of-Words) :**
  + La vectorisation TF-IDF ne tient pas compte de l’ordre des mots ni du contexte dans lequel ils apparaissent.
  + Par conséquent, la nuance et le sens global d’un commentaire peuvent être perdus, notamment dans des phrases où la position des mots est cruciale pour comprendre l’intention.
* **Capacité limitée à capturer les interactions complexes :**
  + La régression logistique, bien qu’efficace pour des problèmes linéaires, peine à modéliser des relations non linéaires entre les termes et à prendre en compte des dépendances complexes entre les différentes classes.
  + Dans une approche OneVsRest, chaque modèle est entraîné indépendamment pour chaque classe, ce qui peut limiter la prise en compte des interactions entre classes.
* **Performances inégales selon les classes :**
  + Les résultats montrent que, bien que le modèle obtienne de bonnes performances pour certaines catégories (comme *toxic* et *obscene*), il échoue sur des étiquettes moins représentées ou plus complexes comme *severe\_toxic*, *threat* ou *identity\_hate*.
  + Par exemple, les rapports de classification indiquent un F1-score particulièrement faible pour ces classes, suggérant que le modèle a du mal à capter les subtilités de ces types de toxicité.

*Résultats du modèle 1 sur le jeu de validation :*

* Précision moyenne : environ 0.87 (pondérée)
* F1-score global : environ 0.68 (micro)
* Faibles performances sur certaines classes sensibles

A screenshot of a computer

AI-generated content may be incorrect.

**Partie 2 : Modèle 2 – Approche Deep Learning avec RNN et LSTM Unidirectionnel**

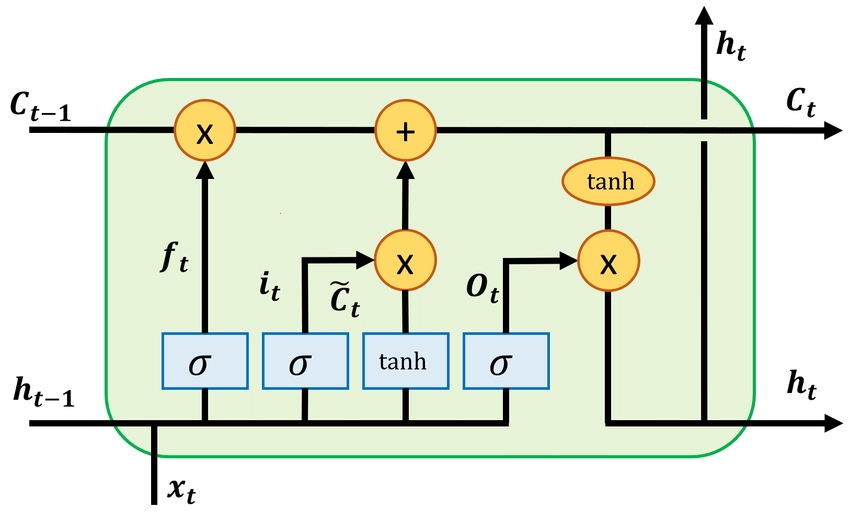
**Présentation du Modèle**

Pour pallier les limitations du modèle 1, le modèle 2 adopte une approche deep learning permettant de mieux capturer le contexte et l’ordre des mots. Pour ce faire, plusieurs étapes de prétraitement et de modélisation sont mises en œuvre :

1. **Nettoyage et Prétraitement du Texte :**  
   Les commentaires sont d’abord nettoyés en les convertissant en minuscules et en supprimant les caractères spéciaux à l’aide d’une fonction de nettoyage (avec re.sub). Ce prétraitement permet d’uniformiser les données et de réduire le bruit.
2. **Tokenisation et Padding :**  
   À l’aide du module Tokenize” de Keras, le texte est converti en séquences d’entiers. On définit un nombre maximal de mots (max\_words = 20000) pour limiter le vocabulaire et une longueur fixe (max\_len = 100) pour les séquences. La fonction pad\_sequences est ensuite utilisée pour s’assurer que toutes les séquences ont la même longueur. Ce processus garantit que les données d’entrée respectent une forme homogène pour le réseau de neurones.
3. **Architecture du Modèle Deep Learning :**  
   Ainsi pour palier au probleme de perte de sens d’une phrase, nous sommes partis sur un modèle de RNN (récurrent neural network) qui lui garde en mémoire la phrase mais le probleme avec un simple RNN c’est qu’il conserve tout en mémoire au début, donc au fil des étapes il peut oublier, ainsi pour palier a ce probleme, nous avons utiliser un LSTM qui resous ce probleme.

Le cœur de l’approche repose sur l’utilisation d’un réseau de neurones récurrent (RNN) intégrant des couches LSTM (Long Short-Term Memory) :

* + **Embedding :**  
    Une couche d’embedding transforme chaque token en un vecteur dense d’une dimension fixée (ici, 128). Cette couche permet d’apprendre une représentation continue des mots, facilitant ainsi la capture des similarités sémantiques.
  + **LSTM Unidirectionnel :**  
    L’architecture inclut deux couches LSTM. La première couche est configurée avec l’option return\_sequences=True, ce qui permet de transmettre l’ensemble de la séquence de sorties à la couche suivante. La deuxième couche LSTM traite ces séquences pour produire un vecteur d’état final. Les LSTM, grâce à leur capacité à mémoriser des informations sur plusieurs pas de temps, permettent de modéliser les dépendances temporelles entre les mots.
  + **Couches Dense et Dropout :**  
    Après les couches LSTM, une couche Dense avec 64 neurones et une activation ReLU est ajoutée pour extraire des caractéristiques supplémentaires. Un Dropout est appliqué ensuite afin de réduire le risque de surapprentissage. Enfin, une couche Dense avec 6 neurones et une activation sigmoid est utilisée pour effectuer la classification multi-label, chaque neurone correspondant à une catégorie de toxicité.



1. **Entraînement et Validation :**  
   Le modèle est entraîné en utilisant la fonction de perte binary\_crossentropy et l’optimiseur adam. Un mécanisme de EarlyStopping surveille la perte de validation pour arrêter l’entraînement lorsque celle-ci cesse de s’améliorer, garantissant ainsi une meilleure généralisation.

**Avantages par Rapport au Modèle 1**

* **Conservation de l’Ordre des Mots :**  
  Contrairement à la méthode TF-IDF qui traite le texte comme un sac de mots, l’utilisation des LSTM permet de conserver l’ordre des mots et de traiter les informations de manière séquentielle. Cela est essentiel pour saisir les nuances contextuelles dans des phrases potentiellement toxiques.
* **Modélisation des Dépendances Contextuelles :**  
  Les LSTM sont conçus pour retenir des informations sur plusieurs pas de temps. Ils peuvent ainsi capturer des relations non linéaires et les dépendances entre les mots, ce qui permet une compréhension plus fine du contenu des commentaires. Par exemple, un LSTM peut tenir compte des termes précédents dans une phrase pour mieux interpréter un mot ambigu ou une expression idiomatique.

**Limitations du Modèle 2**

* **Utilisation d’un LSTM Unidirectionnel :**  
  Le principal inconvénient de ce modèle est qu’un LSTM unidirectionnel lit la séquence dans un seul sens (généralement de gauche à droite). Ainsi, les informations situées à la fin de la séquence peuvent ne pas être suffisamment exploitées, car le modèle ne bénéficie pas du contexte à droite du mot courant.
* **Complexité et Temps d’Entraînement :**  
  Bien que l’approche améliore la capture du contexte par rapport à TF-IDF, l’entraînement de réseaux de neurones récurrents reste gourmand en ressources computationnelles. Cela se traduit par un temps d’entraînement plus long et une nécessité de disposer de matériels performants, surtout lorsque le jeu de données est volumineux.

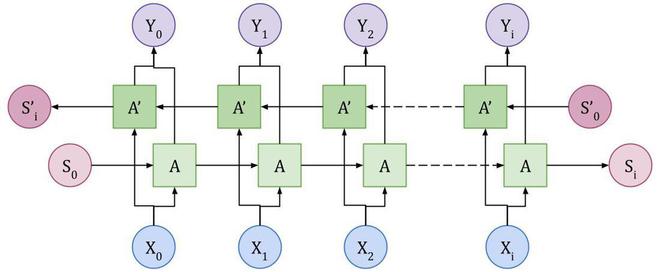
*Résultats du modèle 2 sur le jeu de validation :*

* Amélioration du F1-score pour certaines classes par rapport au modèle 1 (par exemple, une meilleure capture de *toxic* et *obscene*), mais toujours des lacunes notables pour *severe\_toxic*, *threat* et *identity\_hate*.

A screenshot of a table

AI-generated content may be incorrect.

**Partie 3 : Modèle 3 – Approche Optimisée avec LSTM Bidirectionnel et TextVectorization**



**Présentation du Modèle**

Le Modèle 3 représente notre version final en se basant sur l’évolution de nos approches. Il intègre plusieurs améliorations par rapport au Modèle 2 afin de mieux exploiter le contexte des commentaires et d’optimiser le pipeline de traitement. Les principales changements résident dans le prétraitement, la construction du dataset et l’architecture du réseau.

1. **Prétraitement et Vectorisation :**

* **Utilisation de la couche TextVectorization de Keras :**  
  Plutôt que d’utiliser une simple tokenisation suivie de padding, nous utilisons la couche TextVectorization. Celle-ci permet de réaliser en une seule étape le nettoyage (par exemple, mise en minuscules et suppression de caractères spéciaux), la tokenisation et la limitation du nombre de tokens.
  + **Avantage :** Cela simplifie le pipeline de prétraitement et assure une vectorisation cohérente et intégrée.
* **Augmentation du MAX\_FEATURES :**  
  Nous fixons MAX\_FEATURES à 200 000, ce qui permet de constituer un vocabulaire bien plus riche et de couvrir une plus grande partie du lexique utilisé dans les commentaires.
  + **Impact :** Un vocabulaire plus étendu augmente la capacité du modèle à capturer des nuances lexicales importantes.

**2. Création du Dataset avec TensorFlow**

* **Utilisation de l’API tf.data.Dataset :**  
  Les données vectorisées sont encapsulées dans un objet tf.data.Dataset. Cette méthode offre plusieurs avantages :
  + **Caching :** Permet de stocker les données prétraitées en mémoire pour accélérer les itérations.
  + **Shuffling et Batching :** Garantit que le modèle voit des lots de données variés et homogènes, ce qui aide à éviter le surapprentissage.
  + **Prefetching :** Optimise le flux des données pendant l’entraînement en préchargeant les batches.
* **Division en Ensembles :**  
  Le dataset est divisé en trois ensembles (entraînement, validation et test) de manière à assurer un flux de données optimal lors de l’entraînement du modèle.

**3. Architecture du Modèle**

* **Embedding :**  
  Comme pour le modèle précédent, une couche d’embedding convertit les tokens en vecteurs denses. Cette couche apprend une représentation continue des mots, facilitant ainsi la capture des similarités sémantiques.
* **LSTM Bidirectionnel :**  
  La grande innovation ici est l’utilisation d’un LSTM bidirectionnel.
  + **Principe :** Contrairement à un LSTM unidirectionnel qui lit la séquence dans un seul sens (généralement de gauche à droite), le LSTM bidirectionnel lit la séquence dans les deux sens.
  + **Avantage :** Cela permet de mieux saisir le contexte complet d’un commentaire, en intégrant à la fois les informations situées avant et après un mot donné.
* **Couches Denses Supplémentaires :**  
  Après le LSTM bidirectionnel, plusieurs couches denses (128, 256 puis 128 neurones) avec des activations ReLU sont ajoutées.
  + **Objectif :** Ces couches servent à affiner l’extraction des caractéristiques et à améliorer la séparation entre les classes.
* **Sortie Multi-label :**  
  La couche de sortie comporte 6 neurones avec une activation sigmoid, ce qui permet de prédire indépendamment la présence de chaque type de toxicité.
* **Évaluation et Sauvegarde :**
  + **Calcul des Métriques :**  
    Des métriques telles que la précision, le rappel, le F1-score et l’accuracy sont calculées à l’aide des modules Keras et sklearn. Cela permet d’évaluer précisément les performances du modèle sur chaque étiquette et globalement.
  + **Sauvegarde et Rechargement du Modèle :**  
    Le modèle est sauvegardé (au format H5) et rechargé ultérieurement afin de garantir la reproductibilité et de faciliter des tests complémentaires, comme la fonction de scoring d’un commentaire.

*Exemple de résultats obtenus (via Keras et sklearn) :*

**Partie 4 : Axes d'Amélioration et Limites**

**Limites Techniques et Problèmes d'Entraînement**

* **Ressources Computationnelles :**  
  L’entraînement d’un modèle de deep learning, notamment avec un LSTM bidirectionnel et de multiples couches denses, est très exigeant en ressources. Sur Google Colab, les limitations en GPU nous ont contraints à démarrer avec un jeu de données réduit. Une fois satisfaits des premiers résultats, nous avons dû migrer l’entraînement en local, avec des sessions pouvant atteindre 3 heures par epoch, afin d’explorer davantage l’espace des hyperparamètres et de raffiner le modèle.
* **Gestion des Données Déséquilibrées :**  
  Dans la classification multi-label, certaines classes comme *severe\_toxic* et *identity\_hate* sont sous-représentées. Des techniques de rééchantillonnage ou des ajustements de la fonction de perte auraient pu être envisagés pour mieux traiter ce déséquilibre, mais leur implémentation nécessite une approche fine et un volume d’expérimentations conséquent.

**Perspectives d'Amélioration**

* **Augmentation des Données :**  
  L’emploi de techniques d’augmentation de texte (synonymie, paraphrase, ou génération automatique de textes) pourrait enrichir le jeu de données et améliorer la robustesse du modèle en diversifiant les exemples d’entraînement.
* **Modèles Transformer :**  
  Les architectures basées sur des Transformers, telles que BERT ou RoBERTa, offrent une compréhension contextuelle supérieure aux LSTM. Leur capacité à capturer des dépendances complexes et contextuelles pourrait améliorer significativement la classification, bien que leur entraînement demande des ressources computationnelles encore plus importantes.
* **Alternatives aux Embeddings Pré-entraînés :**  
  Nous avons envisagé l’utilisation d’embeddings pré-entraînés comme GloVe. Cependant, ces solutions n’ont pas été retenues pour plusieurs raisons :
  + GloVe ne capture pas de manière contextuelle la signification des mots dans les phrases spécifiques aux commentaires toxiques.
  + Les embeddings contextuels issus des modèles Transformer semblent mieux adaptés pour saisir les nuances du langage, offrant ainsi de meilleures performances sur des tâches complexes de NLP.
* **Optimisation de l'Entraînement :**  
  La mise en place d’un entraînement distribué ou l’utilisation de GPU plus performants pourrait réduire le temps d’entraînement, facilitant ainsi l’exploration de l’espace des hyperparamètres et l’optimisation du modèle.

**Conclusion**

Ce projet de classification des commentaires toxiques a suivi une évolution progressive, en intégrant des approches de plus en plus sophistiquées pour améliorer la capture du contexte et la précision des prédictions.

1. **Modèle 1 (TF-IDF + Régression Logistique)**
   * Approche simple et rapide, permettant d'obtenir un premier benchmark.
   * Bonne interprétabilité, mais des limites importantes dans la compréhension du contexte et des relations entre mots.
2. **Modèle 2 (LSTM Unidirectionnel)**
   * Introduction du deep learning avec des LSTM pour capturer l’ordre des mots et les relations contextuelles.
   * Meilleure compréhension du texte par rapport à TF-IDF, mais unidirectionnalité limitant l’exploitation complète du contexte.
3. **Modèle 3 (LSTM Bidirectionnel + TextVectorization)**
   * Amélioration majeure en intégrant un LSTM bidirectionnel, permettant une lecture du texte dans les deux sens.
   * Pipeline optimisé avec TextVectorization et tf.data.Dataset pour une gestion plus efficace des données.
   * Des couches denses supplémentaires affinant l'extraction des caractéristiques et améliorant la classification multi-label.

**Résultats et Limites**

Le modèle final montre une nette progression en termes de performances, notamment grâce à une meilleure exploitation du contexte des phrases. Cependant, plusieurs défis restent à relever :

* **Problèmes liés aux classes déséquilibrées**, certaines catégories comme *identity\_hate* restant difficiles à détecter.
* **Contraintes computationnelles**, l’entraînement des modèles LSTM nécessitant des ressources conséquentes.

**Perspectives**

Pour aller plus loin, plusieurs pistes d’amélioration peuvent être envisagées :

* **Utilisation de modèles basés sur les Transformers** (ex. BERT, RoBERTa) pour une compréhension encore plus fine du langage.
* **Augmentation des données** avec des techniques comme la reformulation automatique des phrases.
* **Optimisation de l’entraînement** via du calcul distribué ou l’usage de GPU plus performants.

En somme, ce travail a permis d’explorer différentes approches pour la classification de commentaires toxiques, mettant en évidence l’importance de capturer le contexte du langage naturel. L'évolution vers des architectures plus avancées comme les Transformers représente une direction prometteuse pour affiner encore davantage la précision des prédictions.