**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.

*Exemple de résultats du modèle 1 sur le jeu de test :*

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

A screenshot of a computer

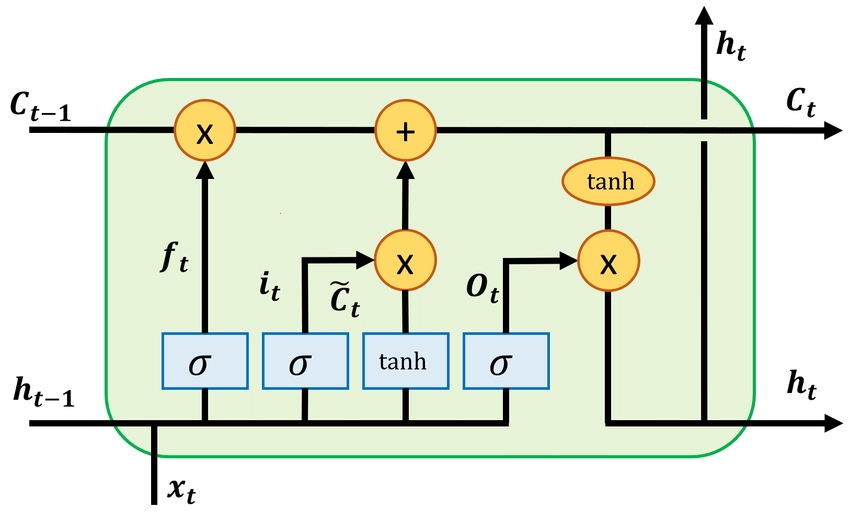
AI-generated content may be incorrect.

**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 (par exemple, avec re.sub). Ce prétraitement permet d’uniformiser les données et de réduire le bruit.
2. **Tokenisation et Padding :**  
   À l’aide du module Tokenizer 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 :**  
   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.
4. **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.

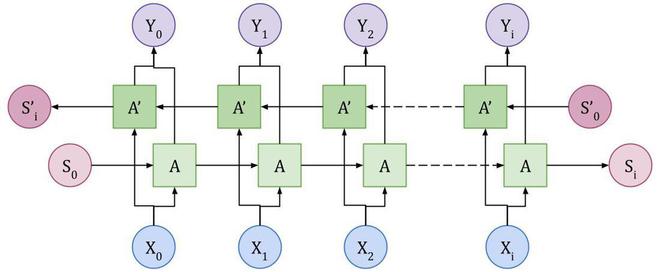
*Exemple de résultats du modèle 2 sur le jeu de validation et de test :*

* 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 l’aboutissement de l’évolution de nos approches. Voici les principaux changements et améliorations apportés par rapport au modèle 2 :

1. **Prétraitement et Vectorisation :**
   * **TextVectorization de Keras :**  
     Le modèle utilise la couche TextVectorization qui permet de transformer directement les textes en séquences numériques en intégrant le nettoyage, la tokenisation et la limitation du nombre de tokens. Cela simplifie le pipeline et assure une vectorisation cohérente.
   * **Augmentation du MAX\_FEATURES :**  
     Un vocabulaire plus riche (jusqu’à 200 000 tokens) est envisagé, garantissant une meilleure couverture du lexique utilisé dans les commentaires.
2. **Création du Dataset avec TensorFlow :**
   * Le dataset est construit à l’aide de l’API tf.data.Dataset, permettant une gestion efficace du caching, du shuffling, du batching et du prefetching.
   * La division du dataset en ensembles d’entraînement, validation et test est réalisée de manière à optimiser le flux de données pour l’entraînement.
3. **Architecture du Modèle :**
   * **Embedding :** Comme pour le modèle 2, une couche d’embedding convertit les tokens en vecteurs denses.
   * **LSTM Bidirectionnel :**  
     L’utilisation d’un LSTM bidirectionnel permet de lire la séquence dans les deux sens (de gauche à droite et de droite à gauche). Cette bidirectionnalité offre une meilleure capture du contexte complet d’un commentaire.
   * **Couches Denses Supplémentaires :**  
     Plusieurs couches denses (avec 128, 256, puis 128 neurones) sont ajoutées après le LSTM pour affiner l’extraction des caractéristiques et améliorer la séparation entre les classes.
   * **Sortie Multi-label :**  
     Une couche de sortie avec 6 neurones et une activation sigmoid permet de prédire la présence de chaque type de toxicité de manière indépendante.
4. **Évaluation et Sauvegarde :**
   * Des métriques précises (précision, rappel, F1-score, accuracy) sont calculées à l’aide des outils Keras et sklearn.
   * Le modèle est sauvegardé et rechargé pour permettre des tests ultérieurs, garantissant ainsi la reproductibilité.

**Résultats et Comparaison**

* **Amélioration Contextuelle :**  
  Le passage à un LSTM bidirectionnel permet une meilleure compréhension du contexte par rapport au modèle 2.
* **Meilleure Généralisation :**  
  La profondeur supplémentaire apportée par les couches denses aide à mieux discriminer entre les classes, même pour celles moins représentées.
* **Performances Globales :**  
  Le modèle 3 montre une amélioration notable des scores (F1, précision et rappel) sur l’ensemble de test, bien que certains problèmes persistent pour les étiquettes très rares.

**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 peuvent ralentir l’entraînement, imposer des restrictions sur la taille des batchs et limiter le nombre d’époques possibles.
* **Optimisation des Hyperparamètres :**  
  L’optimisation fine (nombre d’unités LSTM, taux de dropout, dimensions des couches denses, etc.) reste un défi majeur. Des ajustements supplémentaires pourraient être nécessaires pour atteindre des performances optimales.
* **Gestion des Données Déséquilibrées :**  
  Comme dans beaucoup de tâches de classification multi-label, certaines classes (par exemple *severe\_toxic* et *identity\_hate*) sont moins représentées. Des techniques de rééchantillonnage ou des ajustements de la fonction de perte pourraient être envisagés pour pallier ce déséquilibre.

**Perspectives d'Amélioration**

* **Augmentation des Données :**  
  L’utilisation de techniques d’augmentation de texte pourrait enrichir le jeu de données et améliorer la robustesse du modèle.
* **Modèles Transformer :**  
  Des architectures basées sur des Transformers (comme BERT) pourraient être explorées pour une compréhension contextuelle encore plus fine et des performances supérieures.
* **Optimisation de l'Entraînement :**  
  La mise en place d’un entraînement distribué ou l’utilisation de GPU plus performants permettrait de réduire le temps d’entraînement et d’explorer plus largement l’espace des hyperparamètres.
* **Fine-tuning des Seuils de Décision :**  
  L’ajustement des seuils de décision pour chaque classe pourrait améliorer la précision sur les classes rares.

**Conclusion**

L’évolution des modèles présentés démontre une progression logique et cohérente dans l’approche de la classification de commentaires toxiques. Le modèle 1, basé sur TF-IDF et une régression logistique, constitue une base simple et rapide mais limitée par sa capacité à capter le contexte. Le passage au modèle 2 avec un LSTM unidirectionnel apporte une meilleure modélisation séquentielle, même si la compréhension complète du contexte reste imparfaite. Enfin, le modèle 3, grâce à une vectorisation intégrée et à l’utilisation d’un LSTM bidirectionnel couplé à des couches denses supplémentaires, représente une amélioration significative en termes de capture du contexte et de performances globales, malgré des contraintes liées aux ressources computationnelles et aux données déséquilibrées.