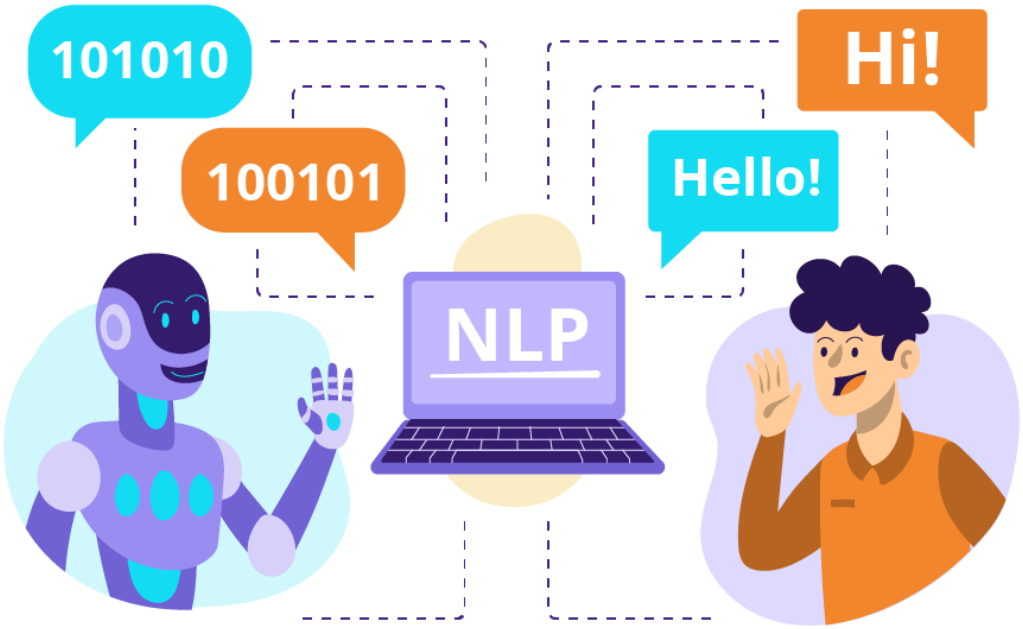
Meurant Edouard
D'Hondt Baptiste

Projet
Natural Language Processing



Toxic comments analyse

15 mars 2024

Dans le cadre de notre cours de NLP, nous avons dû réaliser un projet de classification de commentaire afin de savoir s'ils sont toxiques on non.
Pour cela, un dataset sur le site de Kaggle a été fournis contenant 150 000 commentaires.

Etape 1 : Importation de tous les packages et du jeu de donnée

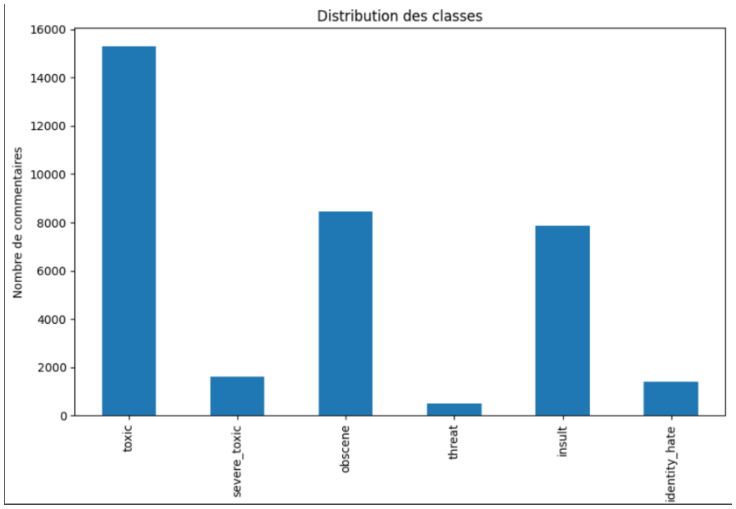
Pour ce projet, nous utiliserons les packages numpy, pandas, tensorflow, nltk, sklearn.

Etape 2 : préparation du jeu de donnée / data cleaning

Pour commencer nous créons des fonctions pour nettoyer tous les commentaires, pour cela, nous enlevons les contractions, les caractères spéciaux, les url, les mentions, les mots trop courts et les mots vides. Dans un second temps, nous remplaçons les abréviations et nous sauvegardons dans un nouveau csv nos commentaires filtrer.

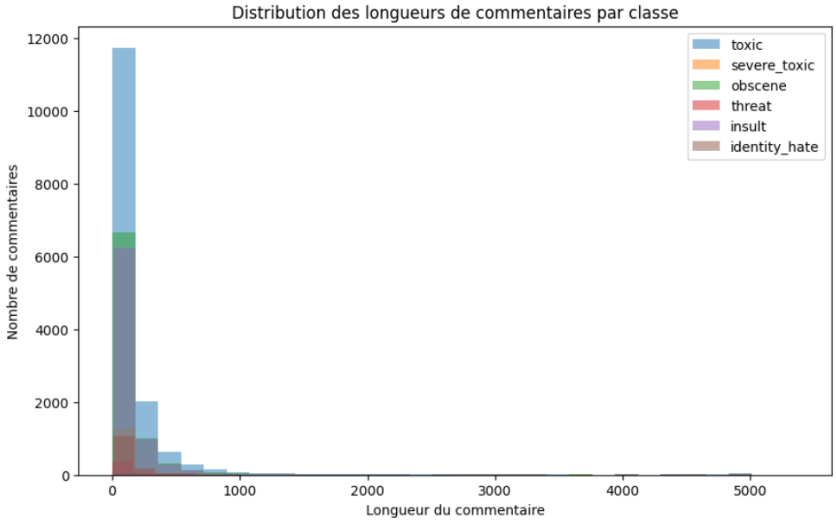
Etape 3 : visualisations des données

Pour visualiser notre jeu de donnée, nous commençons sur un diagramme à barres avec la répartition du nombre de commentaires selon les différentes classes de toxicité. Nous constatons que certaines classes comme "threat" ou "identity_hate" ne comporteront sûrement pas assez de données pour entrainer notre modèle à les repérer.

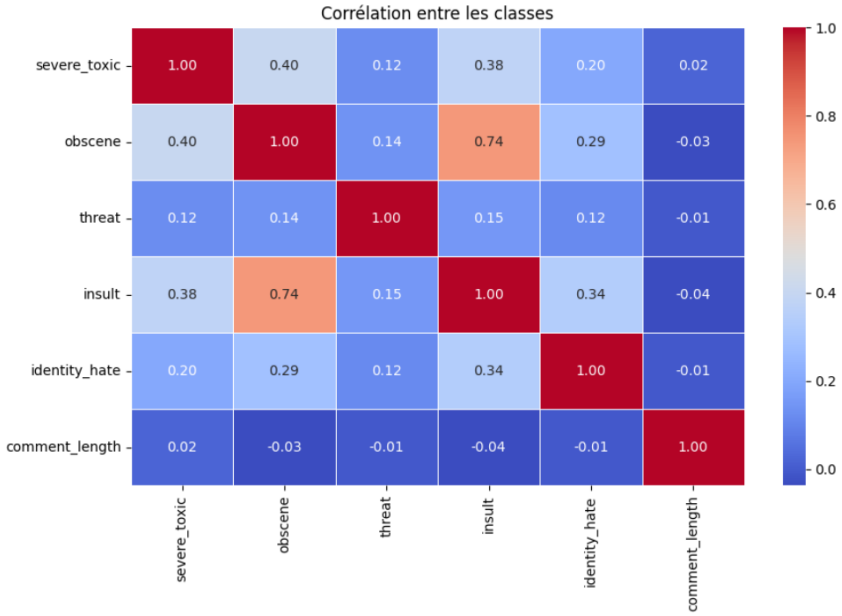


Puis nous faisons un nuage de mot pour chaque classe. Ici, nous prenons l'exemple de la classe severe_toxic.

Une autre donnée qui peut être intéressante est la distribution de la longueur des commentaires. Nous remarquons que la plupart des commentaires ne dépassent pas 100 caractères.



Et pour finir, nous regardons la corrélation entre les différentes classes. On remarque une corrélation importante de 74% entre les commentaires insultants et obscènes.



<u>Etape 4 : création de modèle baseline</u>

Pour commencer la création de nos modèles, nous regroupons toutes les classes en une seul classe « toxique ».

Nous séparons notre dataset en ensemble de données de formation (X_train,Y_train,X_test,Y_test).

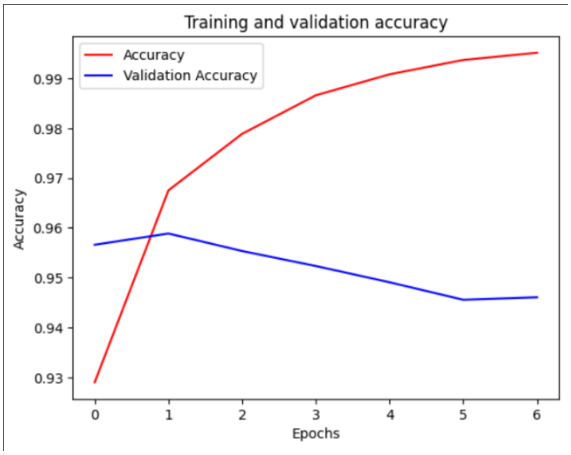
Nous utilisons un TF-IDF sur notre dataset nettoyé pour récupérer les termes les plus importants et récurrents de notre jeu de donnée.

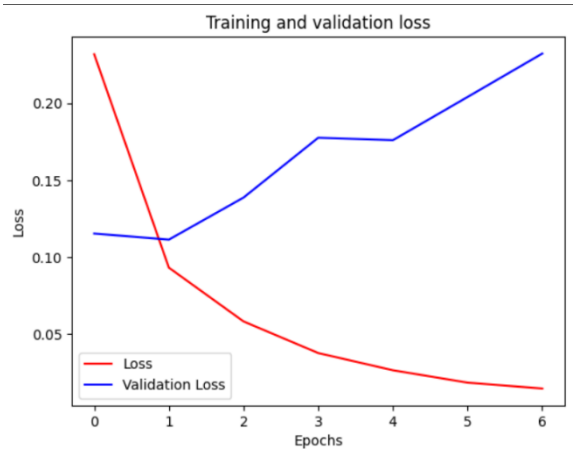
Puis nous entrainons un modèle avec une fonction ramdomforest duquel nous obtenons une accuracy de 0.80 cependant le recall est seulement de 0.59 pour les commentaires toxiques ce

qui n'est pas satisfaisant. Nous entrainons également un modèle avec une fonction de régression logistique qui nous donne des résultats similaires au randomforest.

Etape 5 : création de modèle RNN & LSTM

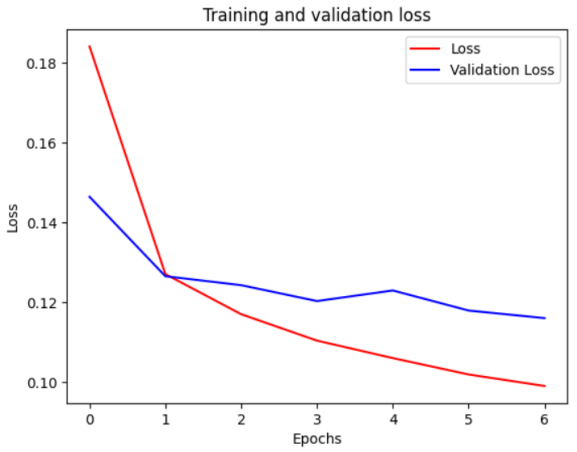
Afin d'obtenir de meilleures performances pour notre modèle, nous essayons d'utiliser un RNN. Il faut commencer par initialiser les paramètres comme l'embedding dimension, la taille maximum des commentaires (ici 50 mots car il s'agit d'un RNN) et le training size égal à la taille de notre corpus. Pour pouvoir entrainer le modèle, nous créons les vecteurs de mapping et d'indices. Nous séparons notre ensemble de données. Enfin, nous définissons un modèle séquentiel LSTM classique avec une couche permettant de capturer les dépendances séquentielles dans les données et une couche de sortie que nous entrainons sur 7 epochs car l'entrainement prend déjà plus d'une heure sur colab. Nous visualisons avec matplot l'entrainement de notre modèle.

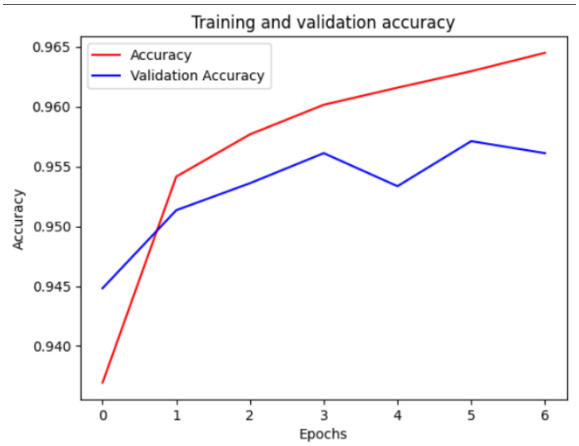




Nous obtenons une accuracy de 0.73 et un recall de 0.77 pour les commentaires toxiques, ce qui est mieux que le random forest. Cependant, les courbes de validation semblent étranges et difficiles à analyser.

Nous allons essayer GloVe pour la création de la matrice d'embedding. Nous réentraînons notre modèle de LSTM et obtenons ceci.



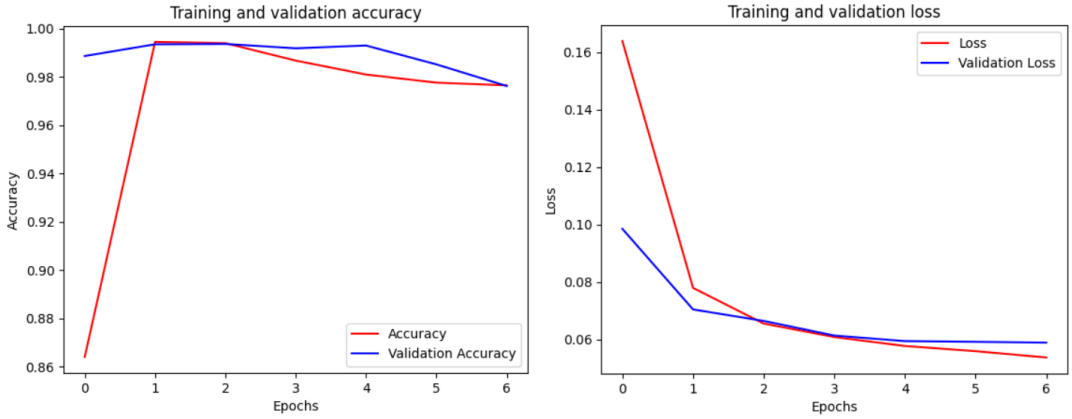


Notre accuracy s'est un peu améliorée avec 0.83 pour les commentaires toxiques mais notre recall a un peu baissé à 0.73. Les courbes paraissent plus logiques mais pas encore optimales.

Ce modèle étant le plus satisfaisant, nous allons maintenant l'entrainer sur les 6 classes du jeu de données à savoir toxic, severe_toxic, obscene, threat, insult et identity_hate.

	precision	recall	f1-score	support
toxic	0.86	0.66	0.75	3276
severe_toxic	0.38	0.02	0.03	327
obscene	0.84	0.66	0.74	1863
threat	0.00	0.00	0.00	101
insult	0.72	0.59	0.65	1702
identity_hate	0.00	0.00	0.00	288

Pour les classes comportant le plus de données, la précision est correcte allant de 72% à 86%. Cependant, le recall n'est pas vraiment satisfaisant allant de 59% à 66%.



Pour cet entrainement, les courbes d'accuracy et de loss sont vraiment belles avec des accuracies de training et validation élevées qui convergent et des losses faibles qui convergent également.

Etape 6 : Pipeline

Afin d'utiliser le modèle, nous avons créé un pipeline dans lequel nous pouvons renseigner une phrase qui sera classée ou non dans les différentes catégories. Voici le résultat pour l'exemple « Disgusting » :

```
toxic: 0.8118398785591125
severe_toxic: 0.017195893451571465
obscene: 0.3821938633918762
threat: 0.011254101991653442
insult: 0.338594913482666
identity_hate: 0.047072965651750565
```

Pour résoudre cet exercice, les réseaux RNN avec embedding aidé de glove semble être la meilleure méthode. Cependant, la structure du modèle doit énormément jouer sur la fiabilité de celui-ci. Pour avoir de meilleurs résultats, il est possible d'optimiser et complexifier le modèle par l'ajout de différentes couches comme des maxpooling, dropout etc.