Projet NLP

Aymeric JAN
Hassan EL MANSOURI KHOUDARI
Mohamed Zineddine CHEDADI

Mai 2021

1 Introduction

Dans le cadre du cours Natural Language Processing du master IASD, nous avons choisi de travailler sur un projet visant à classifier les commentaires dits toxiques en plusieurs catégories.

2 Les données

Les données que nous avons choisi proviennent d'une compétition Kaggle. Le training set est composé d'environ 160 000 commentaires, chacun composé d'un id, du commentaire en question, et de son label décomposé en plusieurs catégories à savoir le commentaire est-il toxique ? Sévèrement toxique ? Obscène ? menaçant ? Insultant ? Ou raciste ?

Name of feature	id	comment text	toxic	severe toxic	obscene	threat	insult	identity hate
type	string	string	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$
feature or label	feature	feature	label	label	label	label	label	label

Figure 1: Structure des données

Nous avons dans un premier temps regarder les corrélations entre les différents labels (figure 2) ainsi que la proportions de chacun des labels au sein du training set (figure 3).

3 Une importante phase de nettoyage des données

Les données étant des commentaires écrits par des utilisateurs, ceux-ci ne sont pas toujours syntaxiquement corrects et utilisent des abréviations ou des expressions. Une première phase consiste donc à nettoyer ces commentaires en les transformant pour les rendre plus homogènes. De plus, on constate que certains de ces commentaires sont accompagnés de l'heure à laquelle ils sont écrit ou de l'adresse IP depuis laquelle ce commentaire a été posté.

1. Nous devons donc retirer dans un premier temps les balises du type "\n"
2. Retirer l'heure ou l'adresse IP à la fin des messages exemple: "D'aww! He matches this background colour I'm seemingly stuck with. Thanks. (talk) 21:51, January 11, 2016 (UTC)"
3. Retirer les éventuels liens hypertextes exemple : "Wikipedia:Good_article_nominations#Transport"

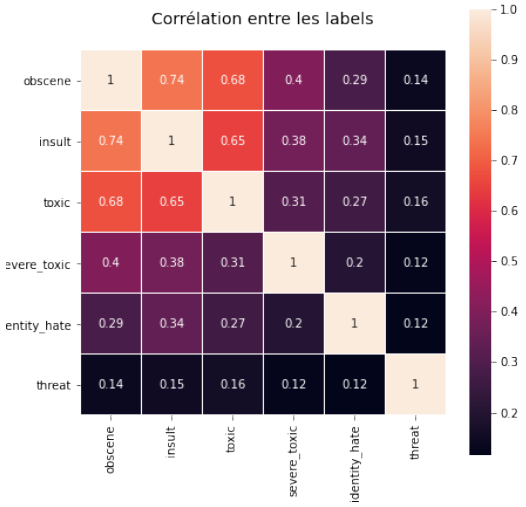

Figure 2: Corrélation entre les labels

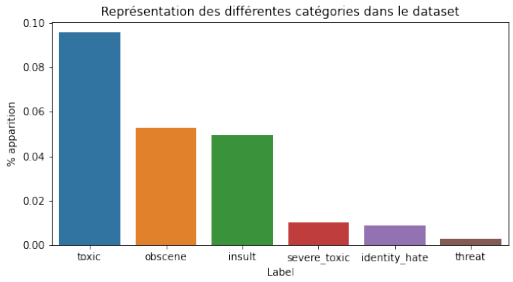

Figure 3: Représentation des différentes catégories dans le dataset

4. Normaliser les commentaires, notamment en enlevant les majuscules bien que les commentaires écrit tout en majuscules peuvent nous donner des indications.
5. Utilisation de fonctions regex pour enlever les abréviations notamment.

4 Les modèles envisagés

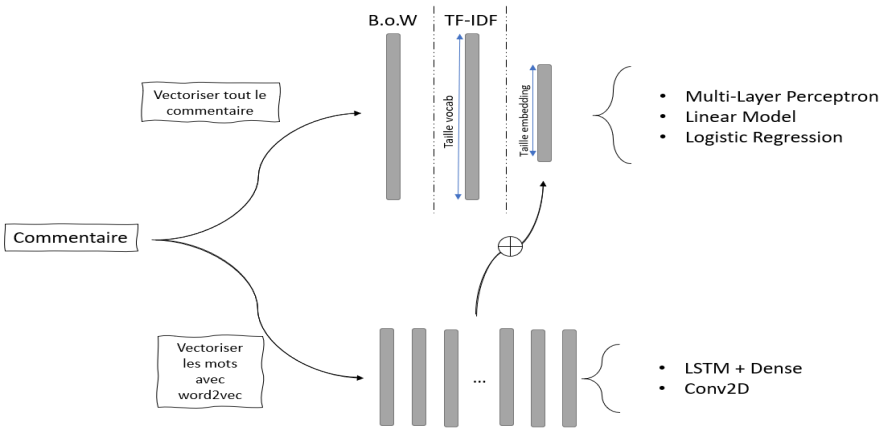

Figure 4: Modèles envisagés

Comme montré sur le schéma ci-dessus (figure 4), nous allons suivre deux manières de faire :

- La première consistera à faire abstraction de l'ordre des mots, soit par une étape d'extraction

d'attributs en utilisant la méthode TF-IDF.
→ Une fois nous avons le vecteur représentatif, nous utilisons des modèles classiques de classifications tel que le MLP, logistic regression, etc.

- La deuxième consistera à prendre en compte l'aspect séquentiel des mots dans les commentaires. On utilise donc un embedding pré-entrainé (word2vec, Glove6B, etc.) pour représenter les mots du commentaire.
→ Nous utiliserons une architecture RNN avec des LSTMs suivie d'une couche dense pour la classification. Nous avons aussi utilisé un modèle de conv2D pour la modélisation.

On note qu'on a utilisé la bibliothèque PyTorch.

5 Méthodologie

Pour les différents modèles que nous allons tester, nous nous accordons sur plusieurs points différents, que ce soit pour la fonction de loss ou pour la méthode d'évaluation des modèles.

5.1 La fonction de loss

Pour les différents modèles implémentés, nous avons choisi d'utiliser comme fonction de loss une *binary cross entropy* à plusieurs composantes: une composantes par catégorie de commentaire toxique. Par ailleurs, comme nous l'avons constaté à l'aide de la figure 3, le dataset ne contient pas de manière équitable tous les labels. En effet, la catégorie la plus représentée parmi les différents commentaire est la catégorie des commentaires dits "toxiques", ils ne représentent cependant que 10 % des échantillons. Pour remédier à cela, nous avons décidé d'utiliser une BCE avec des poids, les poids étant définis de la manière suivante: Soit $Y = (y_i)_{i \leq p} \in \mathbb{R}^{n \times p}$ la matrice contenant les labels où n représente le nombre d'échantillons et p le nombre de catégories, le poids w_i de la catégorie i sera donné par

$$w_i = \frac{1}{\#y_i} \qquad \text{avec } \#y_i = \sum_k^n y_i^{(k)}$$

Ainsi, chacune des catégories sera représentée de la même manière durant l'entraînement, bien que la répartition des labels ne soit pas équitable.

6 Les différentes approches

6.1 TF IDF

Pour cette approche, on teste les performances des modèles qui ne prennent pas en considération l'aspect séquentiel des données. On a commencé par le pré-traitement mentionné dans les sections précédentes. Ensuite on a utilisé la méthode du TF-IDF pour représenter les données textuelles sous forme de vecteur. On a choisi de trois formes de TF-IDF, la première vise les données en terme de mots avec une représentation d'un vecteur de longueur 15.000, la deuxième vise les données en terme de char avec une représentation d'un vecteur de longueur 50.000, et une dernière qui empile les deux précédentes et qui nous donne un vecteur de longueur 65.000. Vu que la troisième méthode performe le mieux, on a décidé de garder que cette représentation dans notre notebook.

Une fois on a eu cette représentation vectorielle, on a utilisé plusieurs méthodes de classification type MLP, logistic regression, XGBoost, Random Forest etc. Pour garder un notebook léger, on a supprimé les implémentations des méthodes les moins performantes (en terme d'accuracy et de généralisation)

et on a gardé XGBoost et logistic regression. Il importe de noter que la classification de nos données se fait label par label. En effet on a entraîner 6 modèles par algorithmes ou chaque modèle classifie un seul label.
On résume les performances de nos deux modèles dans le tableau suivant:

label / modèle	dataset	toxic	severe toxic	obscene	threat	insult	identity hate	mean accuracy
Logistic regression	train	0.969	0.991	0.984	0.997	0.977	0.993	0.985
	test	0.961	0.990	0.979	0.997	0.972	0.992	0.982
XGBoost	train	0.940	0.991	0.977	0.997	0.969	0.993	0.978
	test	0.923	0.994	0.956	0.996	0.954	0.991	0.969

Figure 5: Comparaison des différents modèles au utilisant TF-IDF

On peut déduire du tableau que le modèle le plus performant est la logistic regression avec une moyenne de 98,5% d'accuracy sur le train et 98,2% d'accuracy sur le test.

6.2 CNN

Ce deuxième modèle se base sur les réseaux de neurones convolutifs pour classifier un texte. L'idée est que le résultat de chaque convolution donne une information sur un type de pattern. En faisant varier la taille des noyaux et en concaténant leurs sorties, on se permet de détecter des patterns de tailles multiples (3, 4 ou 5 mots adjacents). Les modèles peuvent être des expressions (ngrammes de mots) Comme "Je déteste", " très bien »et les CNN peuvent donc les identifier dans la phrase quelle que soit leur position.

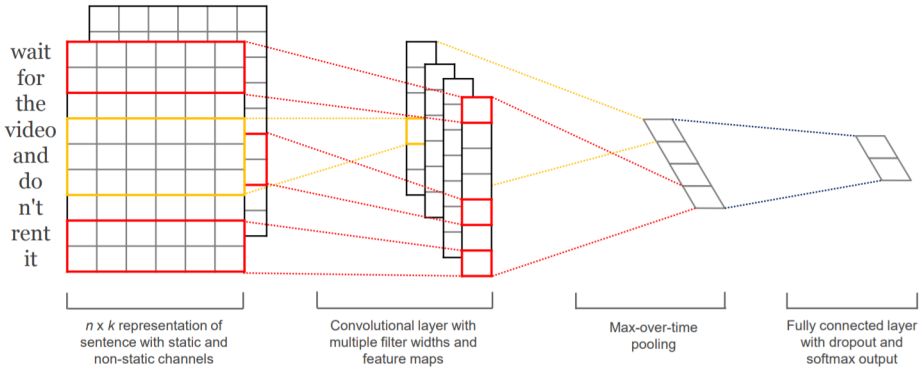

Figure 6: Architecture de modèle avec deux canaux pour une phrase d'exemple

L'idée du modèle est de faire ressortir ces patterns de tailles 3, 4 ou 5. Et puis de prendre le maximum sur une feature map pour ensuite laisser passer la sortie dans une couche complètement connectée qui servira de classifieur. Nous utilisons un embedding Glove de mots pré-entrainés sur twitter de dimensions = 50. Il est judicieux pour nous de l'utiliser parce que l'on a davantage de chance de trouver des mots d'un vocabulaire similaire (acronyme, insultes, etc) dans notre dataset de commentaires.

Pour ce qui est des hyper-paramètres, nous utilisons: des ReLUs, kernel sizes (h) de 3, 4, 5 avec 100 feature maps, un dropout (p) de 0,5, une stride de 3 et un batch size de 32

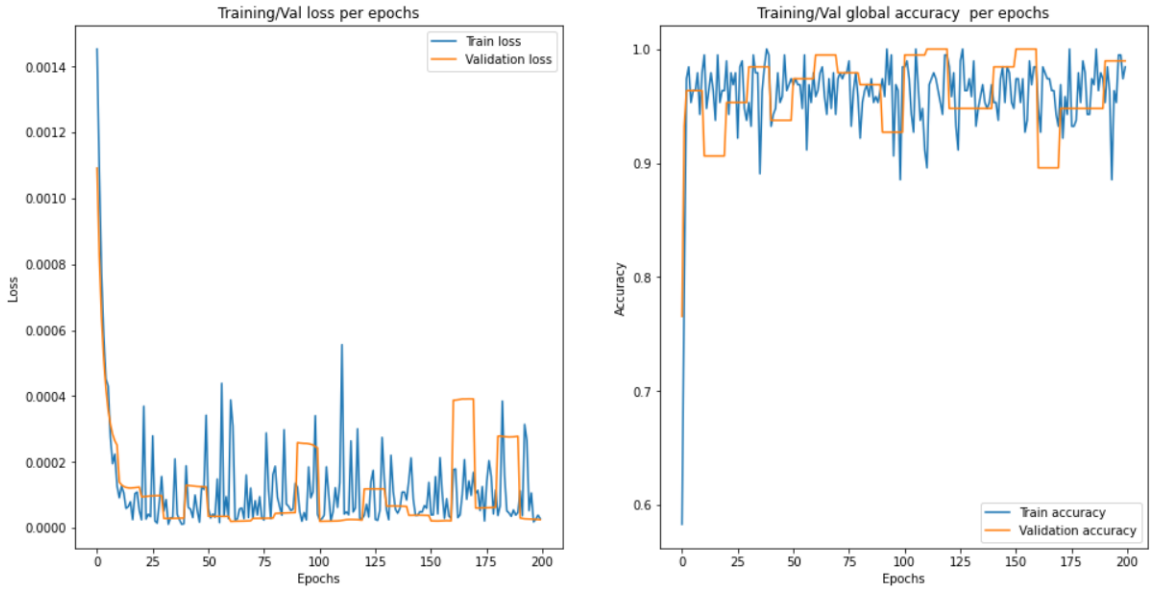

Figure 7: Courbes d'entraînement du modèle CNN

Le modèle converge rapidement vers un minimum, cela se traduit par la chute drastique de la loss dès les premières epochs. Les performances du modèle oscillent après cette chute initiale, si on voulait davantage améliorer le modèle, on pourrait diminuer le learning rate au fur et à mesure des epochs afin de gagner quelques points en précision.

6.3 LSTM

Pour ce dernier modèle, nous avons choisi de nous concentrer sur l'apport que peut nous donner l'insertion de LSTM, c'est à dire l'utilisation d'une architecture récurrente. L'architecture considérée est donc la suivante: une première couche d '*embedding*, un bloc *LSTM* dont on pourra faire varier le nombre de couches, un *max pooling* puis deux couches *fully connected* séparée par une non-linéarité $ReLU$. Par ailleurs, le réseau possède environ 50000 paramètres.

L'embedding est réalisé par Glove6B de dimension 100, c'est à dire que chaque mot est représenté par un vecteur de $\mathbb{R}^{100}$, il n'y a donc pas a ré-entrainé l'embedding.

Le nombre d'échantillon étant assez grand, nous n'avons pas besoin de faire beaucoup d'epochs pendant la phase d'entrainement pour rapidement atteindre des performances plus que remarquable.

Plusieurs combinaisons d'hyperparamètres ont été testés, et deux d'entres elles sont présentées dans les figures qui suivent:

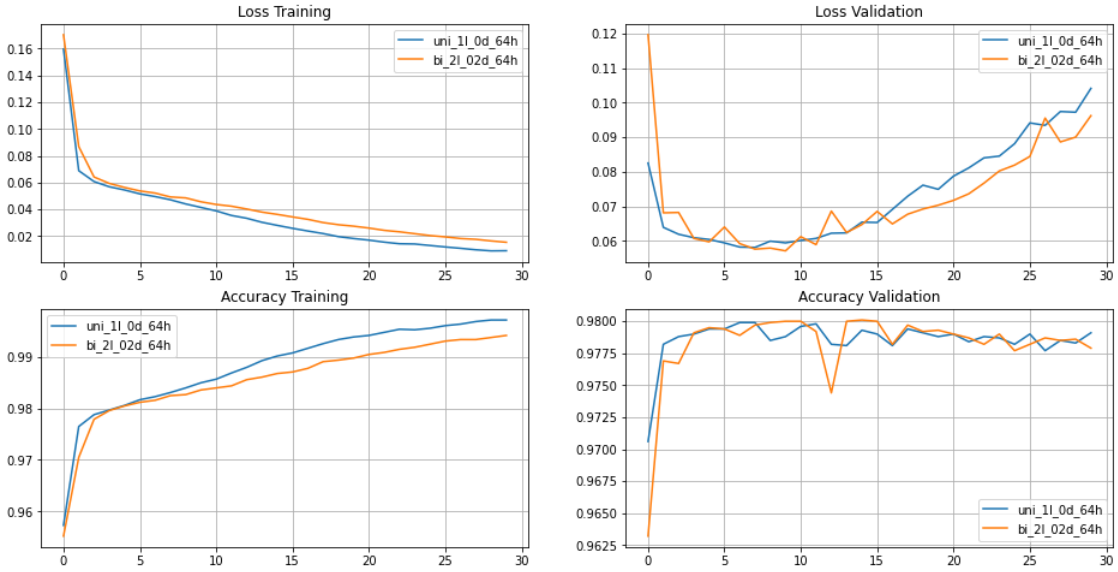

Figure 8: Courbes d'entraînement des modèles LSTM

Le premier modèle nommé *uni_ 1l_ 0d_ 64h* correspond à une architecture dans laquelle on a un LSTM composé d'un seul layer, ayant une dimension de son hidden layer égale à 64 et ne comportant pas de droupout. La deuxième, *bi_ 2l_ 02d_ 64h* correspond à une architecture avec un LSTM bidirectionnel possédant deux layers, avec 20% de dropout et une taille de hidden layers de 64.

On constate d'une part que, dans les deux cas, les performances sont remarquables avec une accuracy de plus de 97.5%, et d'autre part que le second modèle, semble mieux généraliser sur la validation, avec une loss et une accuracy qui sont, certes, moins bonnes en training que celle du premier modèle, mais qui est meilleur en validation car le second modèle à tendance à moins overfitter.

7 Résultats

Pour comparer les différents modèles, nous avons regarder l'accuracy, cette fois-ci par classes, de nos différents modèles sur un ensemble de test, les résultats sont données par le tableau 9

Modèle	toxic	severe toxic	obscene	threat	insult	identity hate
TF IDF	0.9612	0.9903	0.9799	0.9974	0.9728	0.9922
CNN	0.8906	0.9922	0.9609	0.9948	0.9531	0.9844
LSTM	0.9537	0.9880	0.9783	0.9974	0.9686	0.9909

Figure 9: Comparaison des différents modèles au niveau de l'accuracy par catérogie

On constate que les différents modèles choisis arrivent à atteindre des performances très élevées mais de manière assez similaires. Cela peut s'expliquer par le fait que la tâche de classification dans ce cas ci est relativement facile. Cependant, sur des tâches plus complexe comme la prédiction de la fin d'une phrase ou de la traduction, on aurait pu voir une différence dans les performances.

Par ailleurs, les catégories avec les meilleures accuracy sont *identity hate* et *threat*. Cela peut s'expliquer par le fait que certains mots comme des insultes racistes ou des mots en rapport avec la violence peuvent permettre au modèle de déterminer très facilement de quel genre ils relèvent. La toxicité peut, elle, être plus subtile et nécessite de comprendre le contexte de du commentaire en entier.

References

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[2] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722, 2014.

[3] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[4] Martin Sundermeyer, Hermann Ney, and Ralf Schlüter. From feedforward to recurrent lstm neural networks for language modeling. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 23(3):517–529, March 2015.