

# Projet Fil Rouge

□□

## Classification de commentaires toxiques

par

Cherif Ahmed EL ABASS  
&  
Tiphaine Fabre

pour obtenir le diplôme du Mastère spécialisé Expert en Science des Données  
à l'Institut National des Sciences Appliquées de Rouen,  
à soutenir le 1er Juillet 2019 à 9h00.

Encadrant: Stéphane Canu

An electronic version of this project is available at  
[https://github.com/SalamamboBarca/Fil\\_Rouge/](https://github.com/SalamamboBarca/Fil_Rouge/).



# Preface



# Contents

<b>Préface</b>	<b>iii</b>
<b>Introduction</b>	<b>1</b>
<b>1 Présentation des données</b>	<b>3</b>
1.1 Données disponibles . . . . .	3
1.2 Analyse descriptive . . . . .	3
<b>2 Matériel et méthodes</b>	<b>7</b>
2.1 Pré-traitement du texte . . . . .	7
2.1.1 Tokenisation . . . . .	7
2.1.2 Représentation vectorielle . . . . .	7
2.2 Classification des commentaires . . . . .	10
2.2.1 Transformation du problème multi-label . . . . .	10
2.2.2 Réseau de neurones . . . . .	11
2.3 En pratique . . . . .	14
2.3.1 TF-IDF et régression logistique . . . . .	14
2.3.2 Représentation vectorielle et réseau de neurones . . . . .	15
<b>3 Résultats</b>	<b>19</b>
3.1 Comparaison des modèles de représentation vectorielle . . . . .	19
3.2 Discussion . . . . .	20
<b>4 Conclusion</b>	<b>23</b>
<b>Bibliography</b>	<b>27</b>



# Introduction

En vue d'améliorer nos connaissances en Natural Language Programming (NLP) et Machine Learning (ML), nous avons choisi une compétition Kaggle qui a eu lieu en Mars 2018. Cette compétition étant terminée, nous pourrions situer nos résultats dans le classement pour évaluer la performance de notre algorithme de classification.

Le lien vers la compétition est le suivant:

<https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/>

Il s'agit ici d'identifier des commentaires négatifs issus de la plateforme de discussion Wikipédia. Il existe différents types de commentaires négatifs: menaces, obscénités, insultes...

L'objectif est de construire un modèle de classification multiple capable d'identifier les différents types de commentaires toxiques.

ajouter des généralités sur le NLP





# Présentation des données

## 1.1. Données disponibles

Les données mises à disposition sont un large nombre de commentaires Wikipédia étiquetés manuellement par des modérateurs (humains) pour leur caractère toxique. Les différents labels sont:

- toxic
- severe\_toxic
- obscene
- threat
- insult
- identity\_hate

La probabilité qu'un commentaire soit d'un de ces types devra être calculée.  
On dispose de 4 fichiers.

- train.csv - les données d'entraînement, contenant les commentaires avec leur labels binaires
- test.csv - les données tests, ce fichier ne contient pas de labels. Il pourra servir à la prediction pour notre modèle. Les labels ont été ajoutés dans un fichier annexe à la fin de la compétition.
- sample\_submission.csv - un exemple de fichier de soumission au format attendu
- test\_labels.csv - labels pour les données tests, une valeur -1 indique les commentaires qui n'ont pas été utilisés pour le scoring

Le meilleur score obtenu lors de cette compétition est 0.9885. A nous de jouer!

## 1.2. Analyse descriptive

Les données du jeu d'entraînement comportent 159571 observations de 8 variables :

- ID
- Comment Text (le commentaire)
- et les 6 classes citées précédemment (variables binaires indiquant si le commentaire appartient à la classe)

Nous commençons par regarder le nombre d'occurrences de chaque label dans ce jeu d'entraînement.

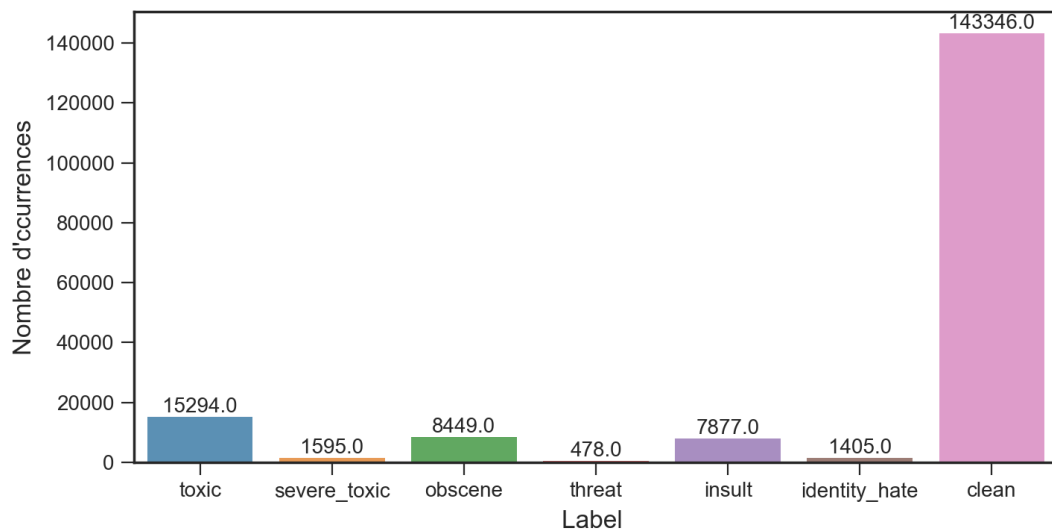


Figure 1.1: Nombre d'occurrences des labels dans le jeu d'entraînement

On voit que pour les données à caractère non injurieux, qu'on qualifiera de "clean" (commentaires n'appartenant à aucune des classes injurieuses), sont beaucoup plus nombreuses (143346) que les autres classes. On remarque aussi que les classes `severe_toxic`, `threat` et `identity_hate` sont quasiment dix fois moins présentes que les autres classes. Par conséquent, on retient que les classes ne sont pas équilibrées dans le jeu de données d'entraînement, ce qu'il ne faudra pas perdre de vue lors de l'évaluation de la qualité de nos modèles.

Maintenant, on s'intéresse aux longueurs des phrases. La longueur des commentaires impacte-t-elle la toxicité de celui-ci ? Pour répondre à cette question, nous avons comparé les longueurs des commentaires "clean" aux autres.

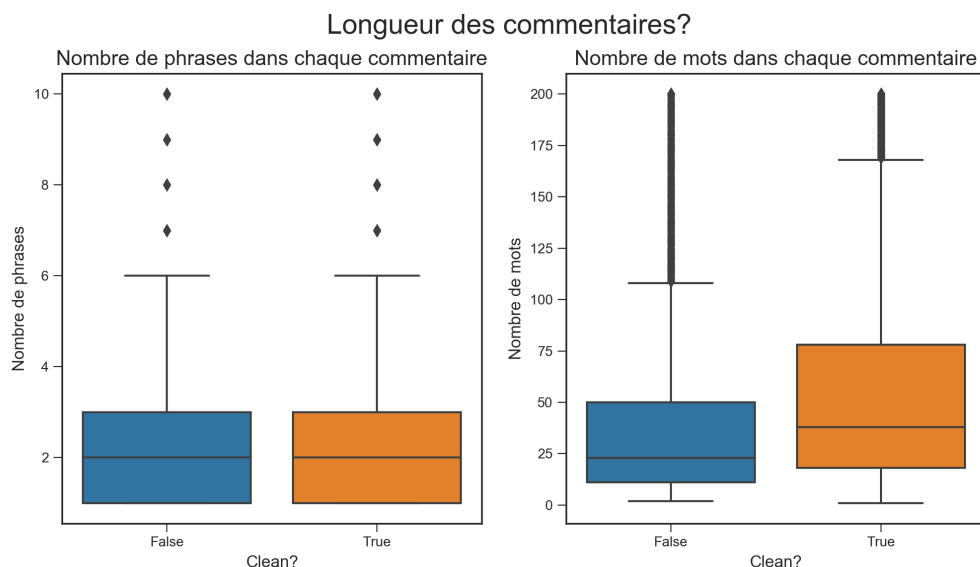


Figure 1.2: Longueur des commentaires du jeu d'entraînement

On voit que le nombre de phrases est quasiment le même pour les commentaires clean et les

autres. Le nombre de mots confirme ce résultat même si l'écart est un peu plus grand. La longueur des commentaires ne semble donc pas avoir d'influence sur son caractère toxique.



# 2

## Matériel et méthodes

### 2.1. Pré-traitement du texte

#### 2.1.1. Tokenisation

Le texte brut représente une longue chaîne de caractères qui peut se révéler complexe à analyser. Pour simplifier sa compréhension, une étape de segmentation est nécessaire. Elle permet une découpe par caractères, mots, phrases etc... Ce processus est nommé tokenisation (tokenization en anglais). En règle générale, les mots sont souvent séparés l'un l'autre par un espace, mais tous les espaces n'ont pas la même signification. Un groupe de mots peut parfois représenter une même unité comme dans les exemples donnés dans la documentation d'IBM<sup>1</sup>. "Los Angeles" et "rock 'n' roll" représentent chacun une unité malgré qu'ils contiennent plusieurs mots et espaces. À l'inverse, deux mots distincts peuvent ne pas être séparés par un espace, comme c'est le cas pour les contractions de langage. "I've" représente les mots "I" et "have". L'espace n'est donc pas toujours un séparateur suffisant.

Selon le problème que l'on se pose, cette étape est très importante. Des erreurs sur le choix du découpage peuvent induire des erreurs d'apprentissage par la suite. Il existe de nombreuses méthodes de tokenisation et il faut souvent faire du sur-mesure pour le jeu de données à étudier.

#### 2.1.2. Représentation vectorielle

Pour notre problème de classification, il est nécessaire de transformer les commentaires (langage naturel) en une représentation que les modèles de machine learning pourront comprendre. En effet, la machine n'étant pas une personne, elle ne comprend pas le langage naturel à son état primaire.

Une représentation récurrente est la représentation de mots dans un espace vectoriel. Les mots ont chacun un vecteur associé. Pourquoi cela? Lorsqu'on voudra comparer des mots entre eux (leur similarité), on calculera une distance. Pour cela, il nous faut une représentation numérique des données. Un vecteur donnera plus d'informations sur le mot qu'un seul scalaire. Voyons ici un exemple simple et souvent repris dans la littérature. Le mot 'man' est très similaire au mot 'woman' dans le sens où ces mots décrivent tous deux des êtres humains. Mais ces deux mots sont souvent considérés comme opposés car ils soulignent une différence au sein de l'espèce humaine. Pour distinguer un homme d'une femme, le modèle a besoin de plus d'une caractéristique. La représentation des mots sous forme vectorielle apparaît alors intéressante.

Il existe différentes méthodes de représentation vectorielle. Nous en avons testé quelques-unes.

##### 2.1.2.1. Word2vec

Word2vec est un ensemble de modèles qui permettent de générer le contexte d'un mot sous forme de vecteur, on parle de word embedding. Ces modèles sont composés de deux réseaux de neurones entraînés pour reconstruire le contexte d'un mot. Le texte, ici nos commentaires, est pris en entrée et Word2vec produit alors un vecteur spatial en sortie. Pour s'entraîner, les modèles ont besoin d'un grand nombre de textes. Cela permet au modèle de mieux généraliser en évitant un apprentissage

<sup>1</sup><https://www.ibm.com/developerworks/community/blogs/nlp/entry/tokenization?lang=en>

trop spécifique.

Le vecteur spatial obtenu a une dimension finie, déterminée à l'avance par l'utilisateur. A chaque mot unique du corpus d'entraînement est attribué un vecteur. Ainsi des mots qui auront un contexte commun seront très proches dans l'espace de représentation des données[2].

Word2vec utilise deux types d'architectures pour la représentation des mots:

- les `continuous bag-of-words` (CBOW), dans lesquels le modèle prédit un mot en fonction des mots de contexte qui l'entourent.
- les `continuous skip-gram`, dans lesquels le modèle utilise un mot pour prédire les mots de contexte qui l'entourent (voir la figure 2.1).

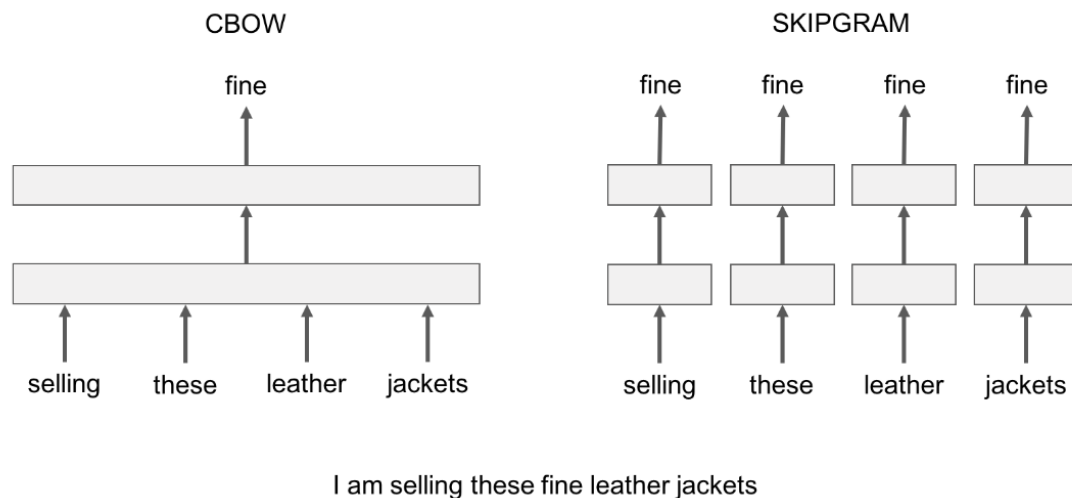


Figure 2.1: Schéma explicatif des architectures CBOW et skip gram

L'architecture skip-gram est plus performante que CBOW pour des jeux de données de grande taille ou des dimensions de vecteurs élevées. En revanche CBOW est moins coûteux en temps de calcul[2].

La paramétrisation des modèles Word2vec influence beaucoup l'apprentissage et quelques paramètres sont particulièrement essentiels:

- les `fonctions d'optimisation`  
Deux fonctions sont disponibles. Le softmax hiérarchique (HS) et l'échantillonnage négatif (NS). Avec la méthode HS, le modèle maximise la log-vraisemblance alors qu'avec la méthode NS minimise la log-vraisemblance des instances négatives. HS fonctionne mieux pour les mots de faible fréquence au contraire de NS. De plus, plus le nombre d'époques est élevé, moins HS est efficace.
- la `dimension`  
Une dimension élevée du vecteur spatial garantit une meilleure qualité de word embedding, jusqu'à un certain point...Au-delà d'une certaine dimension, la qualité de la représentation diminue. L'intervalle idéal est compris entre 100 et 1000.
- la `fenêtre de contexte`  
Elle détermine le nombre de mots avant et après un mot donné à considérer comme contexte.

De façon générale, plus le nombre de mots dans le jeu de données augmente, plus le modèle est performant.

### 2.1.2.2. Glove

GloVe, comme Word2vec, permet de représenter les mots dans un espace vectoriel de façon non supervisée. La distance entre les mots dans l'espace représente leur similarité sémantique. A la différence de Word2vec, le modèle GloVe ne prend pas en entrée des phrases mais une matrice de

co-occurrence entre mots présent dans le corpus d'entraînement. [3] Cette matrice attribue la fréquence à laquelle deux mots apparaissent l'un après l'autre dans le corpus. Une seule lecture du corpus est nécessaire pour obtenir ces statistiques. Cette lecture peut se révéler coûteuse lorsque la taille du corpus est importante mais une fois qu'elle est réalisée, on a pas besoin d'y revenir. Celle-ci permet également de gagner du temps sur la deuxième partie de l'entraînement car le nombre de co-occurrences non nulle est beaucoup plus faible que le nombre total de mots présents dans le corpus[? ].

GloVe utilise la méthode des moindres carrés pondérés. Ce modèle se base sur l'intuition suivante: la probabilité de co-occurrence de deux mots reflète une forme de sémantique. Si on reprend l'exemple donné par les créateurs de GloVe <sup>2</sup>, on peut voir dans la figure 2.2 que le mot 'ice'<sup>3</sup> apparaît plus souvent avec 'solid' qu'avec 'gas' alors que le mot 'steam'<sup>4</sup> apparaît plus souvent avec 'gas' qu'avec 'solid'.

Probability and Ratio	$k = \text{solid}$	$k = \text{gas}$	$k = \text{water}$	$k = \text{fashion}$
$P(k \text{ice})$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k \text{steam})$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k \text{ice})/P(k \text{steam})$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

Figure 2.2: Exemples de probabilités de co-occurrences

En revanche, ces deux mots ont des probabilités proches pour les mots 'water' et 'fashion'. Ils partagent une propriété commune qu'est l'eau et apparaissent tout deux de façon peu fréquente avec le mot 'fashion'. Le ratio des probabilités permet d'établir si un mot co-occurent est plus spécifique d'un mot que de l'autre. Ici, un ratio très supérieur à 1 signifie que le mot  $k$  est associé avec 'ice' et un ratio très inférieur à 1 signifie que le mot  $k$  est associé avec 'steam'. Les ratios proches de 1 ne permettent pas de faire ressortir une préférence.

GloVe cherche à optimiser ses vecteurs de mots de façon à ce que leur produit scalaire soit égal au logarithme de la probabilité de co-occurrence des mots. Le logarithme d'un ratio étant la différences des logarithmes, l'objectif d'optimisation associe le ratio de co-occurrence à la différence de vecteurs dans l'espace vectoriel de représentation. Cela permet aux vecteurs de mieux représenter l'analogie de certains mots.

### 2.1.2.3. Fasttext

Développé par le 'Facebook's AI Research (FAIR) lab', fastText est une extension des modèles Word2vec. De la même façon, cet outil utilise les architectures de CBOW<sup>5</sup> et de skip grams[1]. La différence entre fastText et Word2vec (ou GloVe) est la suivante. Word2vec traite chaque mot comme une entité atomique du corpus d'entraînement et génère un vecteur pour chaque mot. En revanche, fastText traite chaque mot comme une aggrégation de n-grams caractères. Le vecteur d'un mot est alors la somme des n-grams caractères. Par exemple, le vecteur du mot 'usage' est la somme des vecteurs des n-grams "usa", "sag", "ag", "usag", "sage" et "usage" si l'on choisit de considérer les n-grams de longueur 3 à 5.

Cette méthode génère de meilleurs contextes (word embeddings) pour les mots rares. Même si le mot est rare, ces ngrams se retrouvent dans d'autres mots<sup>6</sup>. De plus, fastText peut créer le vecteur d'un mot à partir de ngrams même si le mot n'apparaît pas dans le corpus d'entraînement. Word2vec et Glove en sont incapables. Le choix de la taille minimum et maximum des n-grams à considérer est essentielle et rallonge d'autant plus le temps d'entraînement du modèle.

### 2.1.2.4. TFIDF

Au delà de la représentation vectorielle de mots, il existe d'autres méthodes pour représenter le texte.

<sup>2</sup><https://nlp.stanford.edu/projects/glove/>

<sup>3</sup>ice signifie glace.

<sup>4</sup>steam signifie vapeur, buée.

<sup>5</sup>CBOW: continuous bag-of-words

<sup>6</sup><https://fasttext.cc/docs/en/support.html>

La méthode pondérée du TF-IDF (de l'anglais term frequency-inverse document frequency) permet d'évaluer l'importance d'un mot contenu dans un document (ici un commentaire), relativement à un corpus (ici notre ensemble de commentaires). Le principe est le suivant. Le poids attribué à chaque mot est proportionnel au nombre d'occurrences de ce mot dans le document (TF pour term frequency). Il varie également en fonction de la fréquence du mot dans le corpus (IDF pour inverse document frequency). En effet, les mots apparaissant dans tous les documents (tels les articles définis - le, la, les) sont peu discriminant. C'est pourquoi la pertinence d'un mot augmente en fonction de sa rareté au sein du corpus.

Le poids TF-IDF est en fait le produit des deux fréquences:

$$\text{tfidf}_{i,j} = \text{tf}_{i,j} \cdot \text{idf}_i$$

avec  $\text{tf}_{i,j}$  la fréquence brute d'un mot  $i$  donné, soit le nombre d'occurrences de ce mot dans le document  $j$  considéré. De nombreuses variantes existent pour le calcul de cette fréquence.

et avec  $\text{idf}_i$  la fréquence inverse de document qui mesure l'importance du mot  $i$  dans l'ensemble du corpus.

$\text{tf}_{i,j}$  est calculée comme suit:

$$\text{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

$\text{idf}_i$  consiste à calculer le logarithme (en base 10 ou en base 21) de l'inverse de la proportion de documents du corpus qui contiennent le mot :

$$\text{idf}_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|} \quad \text{idf}_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

où :

$|D|$  est le nombre total de documents dans le corpus

$|\{d_j : t_i \in d_j\}|$  est le nombre de documents où le terme  $t_i$  apparaît<sup>7</sup>.

## 2.2. Classification des commentaires

Une fois la transformation du texte faite, on peut passer à la classification du texte.

### 2.2.1. Transformation du problème multi-label

Lorsqu'on traite un problème multi-label, on veut pouvoir mettre un commentaire dans plusieurs cases. Pour les modèles de machine learning, il est plus facile de traiter un problème multi-classe, qui attribuera un label par commentaire. On peut alors opérer une transformation du problème en considérant chaque label indépendamment. Plusieurs options s'offrent à nous:

- **la classification binaire**  
C'est probablement la méthode la plus simple, qui traite chaque label séparément. On assume ici qu'il n'existe pas de corrélation entre les différents labels.
- **la classification en chaîne**  
On entraîne d'abord le premier modèle sur le jeu d'entraînement puis les suivants sont entraînés à la fois sur le jeu d'entraînement mais également sur les prédictions des modèles précédents.
- **le label Powerset** La méthode de "label Powerset" consiste à considérer chaque combinaison de labels possible comme une classe à part entière. Dans notre cas, nous avons 6 labels donc 63 combinaisons possibles.

Grâce au deep learning, la transformation du problème ne devient plus nécessaire. Les réseaux de neurones sont capables d'attribuer plusieurs labels à un commentaire.

<sup>7</sup>formules issues de <https://fr.wikipedia.org/wiki/TF-IDF>



## 2.2.2. Réseau de neurones

### 2.2.2.1. Rappels sur les réseaux de neurones

L'architecture des réseaux de neurones s'inspire du fonctionnement des neurones biologiques. Cette modélisation constitue les neurones formels. Ils peuvent théoriquement réaliser des fonctions logiques, arithmétiques et symboliques complexes. Par exemple, un neurone peut sommer ses entrées, comparer la somme résultante à une valeur seuil, et répondre en émettant un signal si cette somme est supérieure ou égale à ce seuil. Une fois ces neurones associés entre eux, ils forment un réseau dont les connexions sont variables (*convolutifs, récurrents...*). La transmission de signaux d'un neurone à l'autre se fait via des poids synaptiques qui peuvent être modulés par des règles d'apprentissage. La variation de ces poids permet aux réseaux d'améliorer l'exécution de la tâche qui lui est confiée. Pour un problème de classification comme le notre, le réseau fera varier ces poids jusqu'à ce qu'il classe chaque commentaire avec une erreur minimale (en se basant sur les labels de chaque commentaire en apprentissage supervisé). C'est un problème d'optimisation.

Le réseau de neurones est composé d'une succession de couches dont chacune prend en entrée les sorties de la couche précédente. Chaque couche ( $i$ ) est composée de  $N_i$  neurones, prenant leurs entrées sur les  $N_i - 1$  neurones de la couche précédente. Ces entrées sont alors combinées via la fonction de combinaison (voir la figure 2.3). Cette fonction produit une valeur scalaire, on parle également de fonction vecteur-à-scalaire. Il en existe plusieurs variantes:

- les *multi-layer perceptrons* (MLP), qui calculent une combinaison linéaire des entrées. La fonction de combinaison renvoie le produit scalaire entre le vecteur des entrées ( $x_1, \dots, x_n$ ) et le vecteur des poids synaptiques ( $w_{1i}, \dots, w_{ni}$ ).
- les *radial basis functions* (RBF), qui calculent la distance entre les entrées. La fonction de combinaison renvoie la norme euclidienne du vecteur issue de la différence vectorielle entre les vecteurs d'entrées.

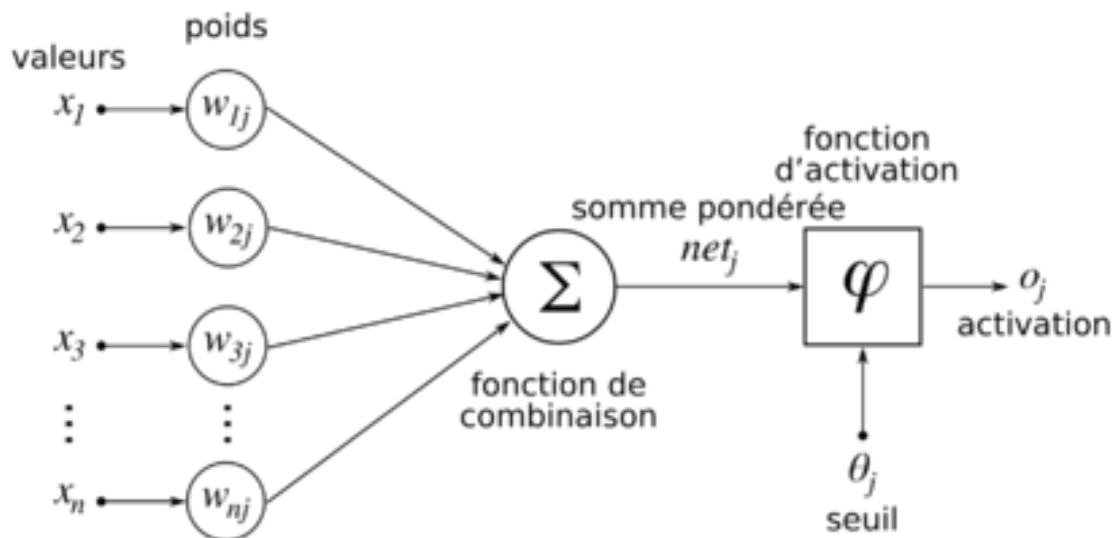


Figure 2.3: Structure d'un neurone artificiel. Le neurone calcule la somme de ses entrées puis cette valeur passe à travers la fonction d'activation pour produire sa sortie. Source: [https://fr.wikipedia.org/wiki/Réseau\\_de\\_neurones\\_artificiels](https://fr.wikipedia.org/wiki/Réseau_de_neurones_artificiels)

Pour produire une sortie non-linéaire, les neurones ne sont pas toujours activés. Ce comportement est dû à la fonction d'activation (ou fonction de seuillage). Lorsque le scalaire produit est en dessous du seuil, le neurone est non-actif. Au dessus du seuil, le neurone est actif. Aux alentours du seuil, on parle de phase de transition. Il existe de nombreuses fonctions d'activation (e.g la fonction sigmoïde ou tangente hyperbolique). De plus, lors de l'apprentissage, le réseau de neurone peut minimiser son erreur grâce à la rétropropagation. Il s'agit de rétropropager l'erreur commise par un neurone à ses synapses et aux neurones qui y sont reliés. Les poids synaptiques qui contribuent à une erreur importante sont modifiés de manière plus significative que les autres.

### 2.2.2.2. Réseau Neural Récurrent

Les réseaux de neurones récurrents (RNN<sup>8</sup>) sont constitués de neurones interconnectés. Il existe au moins un cycle dans leur architecture, on parle de connexion récurrente. Leur principale intérêt réside dans le fait qu'ils sont adaptés à l'analyse de séquences de taille variable. Leur utilisation va de la reconnaissance automatique de formes (parole ou écriture manuscrite) à la traduction automatique. Ils se révèlent donc intéressants pour des tâches de NLP<sup>9</sup>. Si on les déplie, ils deviennent comparables à des réseaux de neurones classiques avec des contraintes d'égalité entre les poids du réseau (voir la figure 2.4). Les poids  $U$ ,  $V$  et  $W$  ne varient pas au cours du cycle. Les RNNs s'entraînent de la

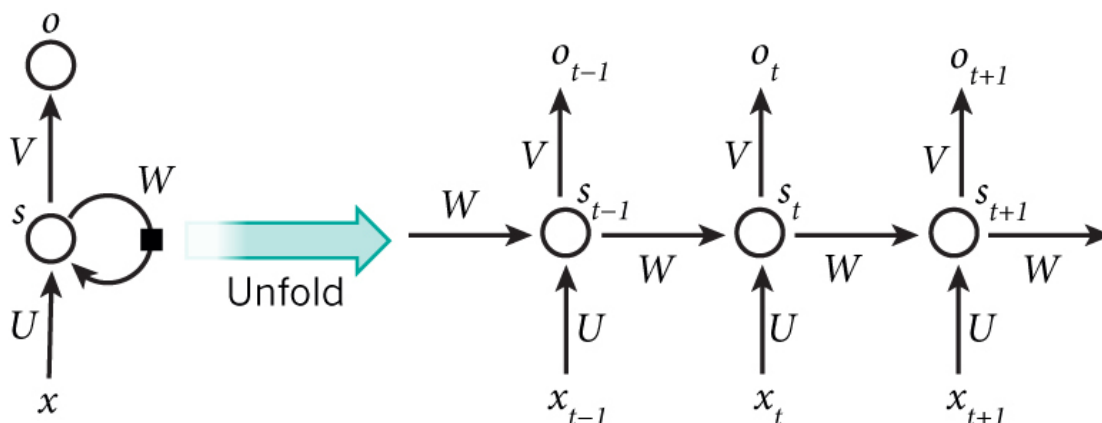


Figure 2.4: Schéma d'un réseau de neurones récurrents à une unité reliant l'entrée et la sortie du réseau. À droite la version « dépliée » de la structure. Source: <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>

même façon que les réseaux classiques en utilisant la rétropropagation du gradient mais font face au problème de disparition du gradient ("vanishing gradient" en anglais) lorsqu'ils mémorisent des événements passés. L'architecture des réseaux de type Long short-term memory (LSTM) palie à ce problème de gradient.

Les neurones de type LSTM possèdent une mémoire des observations précédentes. Ils sont composés d'une cellule et trois portails: entrée, sortie et oubli. La cellule retient des valeurs à intervalle de temps aléatoire et les portails régulent l'information arrivant à la cellule (voir la figure 2.5).

Pour améliorer la performance du réseau récurrent, un apport supplémentaire d'information en entrée du réseau peut être bénéfique. Cela est possible grâce aux réseaux bidirectionnels (BRNNs<sup>10</sup>). Ils connectent deux couches cachées de direction opposée vers une même sortie. Ainsi la couche de sortie peut recevoir des informations des observations passées et futures de façon simultanée[4]. Ces réseaux sont particulièrement efficaces lorsque le contexte d'une entrée doit être pris en compte. Par exemple, dans la reconnaissance d'écriture, il est intéressant de connaître les lettres situées avant et après la lettre qu'on observe.

### 2.2.2.3. Evaluation de la qualité du modèle

Il existe différentes mesures pour évaluer la qualité d'un modèle. La plus populaire est l'*accuracy* mais elle n'est pas toujours la plus pertinente en fonction du problème à résoudre. Elle mesure le pourcentage de réponses correctement prédites par le modèle. Notamment, dans un problème de classification où la distribution des classes n'est pas uniforme, l'*accuracy* peut encourager un modèle qui en réalité ne donne pas une bonne classification.

Soit la matrice de confusion:

	Classe 0 prédite	Classe 1 prédite
Classe 0 observée	$C_{0,0}$	$C_{0,1}$
Classe 1 observée	$C_{1,0}$	$C_{1,1}$

<sup>8</sup>Recurrent Neural Network en anglais

<sup>9</sup>Natural Language Processing

<sup>10</sup>Bidirectional Recurrent Neural Networks

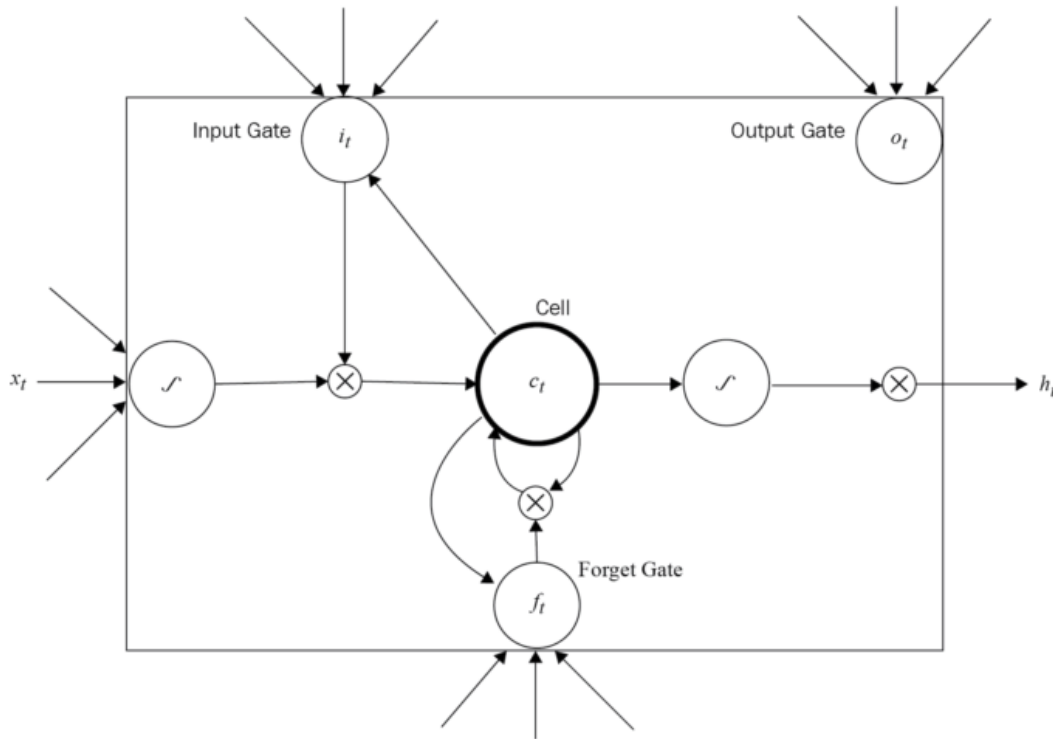


Figure 2.5: Schéma d'une unité LSTM. Source: <https://hub.packtpub.com/what-is-lstm/>

L'accuracy est un ratio des éléments de cette matrice:

$$\text{acc} = \frac{C_{0,0} + C_{1,1}}{C_{0,0} + C_{1,1} + C_{0,1} + C_{1,0}}$$

Imaginons que la classe 0 est beaucoup plus représentée que la classe 1 (50000 contre 50 individus). Le modèle aura souvent tendance à prédire la classe 0 plutôt que la classe 1 car il aura eu plus d'exemples représentant la classe 0. Donc  $C_{0,0}$  sera très élevé et les autres valeurs seront faibles. L'accuracy reviendra à calculer  $C_{0,0}/(C_{0,0} + \alpha)$  avec  $\alpha$  une toute petite valeur. L'accuracy sera alors proche de 1 mais le modèle sera mauvais car il ne prédira jamais la classe 1, rare.

Pour palier à ce phénomène, d'autres mesures existent<sup>11</sup>. Elles utilisent également les valeurs de la matrice de confusion. Celle-ci donne des informations intéressantes sur les prédictions effectuées. Les éléments de la matrice de confusion  $C$  correspondent au nombre d'observations connues pour appartenir au groupe  $i$  mais prédites dans le groupe  $j$ . Dans notre cas, le groupe 1 est le label pour lequel on a entraîné le modèle et 0, les autres. On peut mesurer le `recall` ou la `precision`. Ces mesures permettent d'identifier des problèmes de déséquilibre au sein des données.

Le `recall` d'une classe est défini par le ratio du nombre d'exemples classés correctement pour cette classe sur le nombre total d'observations appartenant à cette classe. Pour la classe 0, le `recall` est le suivant:

$$\text{recall}(0) = \frac{C_{0,0}}{C_{0,1} + C_{0,0}}$$

Un `recall` élevé indique un nombre de faux négatifs faible ( $C_{0,1}$ ). La précision d'une classe est elle définie par le ratio du nombre d'exemples classés correctement pour cette classe sur le nombre total

<sup>11</sup><https://www.geeksforgeeks.org/confusion-matrix-machine-learning/>

de prédictions de cette classe. Pour la classe 0, la précision est la suivante:

$$\text{precision}(0) = \frac{C_{0,0}}{C_{1,0} + C_{0,0}}$$

Une précision élevée indique un nombre de faux positifs faibles ( $C_{1,0}$ ).

Ainsi, un fort recall et une faible précision signifient que la majorité de la classe 0 est correctement prédite mais il y a beaucoup de faux positifs. A l'inverse un faible recall et une forte précision indiquent que le modèle prédit beaucoup de faux négatifs mais les vrais positifs sont importants.

Pour étudier ce ratio entre les deux mesures, on calcule la F-mesure, une moyenne harmonique<sup>12</sup> qui pénalise les valeurs extrêmes. Sa valeur se rapproche toujours de la valeur minimale entre précision et recall.

$$F - \text{mesure} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

## 2.3. En pratique

Comme nous l'avons vu précédemment, il existe de nombreuses méthodes de représentation du texte. Nous avons testé les 4 méthodes présentées dans la partie 2.1.2

### 2.3.1. TF-IDF et régression logistique

Pour une première manipulation des données, nous avons décidé d'utiliser des concepts simples. Un premier notebook `log_reg.ipynb` a été réalisé en utilisant le TF-IDF et la régression logistique comme méthode de classification.

Les notebooks du projet contiennent des commentaires et des liens vers la documentation des packages utilisés pour une aide à la compréhension.

Dans ce premier notebook, on observe différentes étapes.

#### 1. le nettoyage du texte

Cette étape est optionnelle. Elle consiste à uniformiser le texte d'entrée. On peut choisir de gommer les erreurs de frappe, remplacer des caractères particuliers, retirer la ponctuation ou bien encore dans notre cas, retirer les contractions de langage (e.g. "I've been" sera remplacé par "I have been"). Cette étape est souvent utile pour améliorer la tokenisation du texte.

#### 2. la tokenisation

Ici, on choisit l'espace comme séparateur.

#### 3. la vectorisation

L'outil `TfidfVectorizer` développé dans scikit-learn permet la création d'une matrice TF-IDF. Les lignes représentent les commentaires disponibles dans le jeu de données et les colonnes représentent les mots qu'on considère comme intéressants. Ici, on a choisi de garder uniquement les 5000 termes les plus fréquents dans le corpus grâce à l'option `max-features` sans considérer les petits mots fréquents (e.g le, la, les) avec l'option `stop_words`. On applique cet outil sur les jeux d'entraînement et de test (voir la figure 2.6).

#### 4. la classification

On utilise ici la classification binaire (voir la partie 2.2.1). Une régression logistique est effectuée pour chacun des 6 labels de façon indépendante.

#### 5. la vérification du modèle

On mesure ici l'accuracy avec la fonction `accuracy_score` développée dans sklearn et la matrice de confusion (voir la partie 2.2.3).

<sup>12</sup>[https://fr.wikipedia.org/wiki/Moyenne\\_harmonique](https://fr.wikipedia.org/wiki/Moyenne_harmonique)

```
In [7]: print('Dimension de la matrice document-terme d\'entraînement')
print(x_train_tfidf.shape)
print('\nDimension de la matrice document-terme test')
print(x_test_tfidf.shape)

Dimension de la matrice document-terme d'entraînement
(159571, 5000)

Dimension de la matrice document-terme test
(153164, 5000)
```

Figure 2.6: Dimensions des matrices TF-IDF obtenues à partir des jeux d'entraînement et test.

## 2.3.2. Représentation vectorielle et réseau de neurones

Dans les notebooks suivants, nous avons voulu comparer les différents modèles vecteur pour un réseau de neurones de même structure inspiré par le notebook Kaggle de Jeremy Howard<sup>13</sup>.

Nous avons suivi la même ligne conductrice que dans la partie précédente, à savoir le nettoyage, la tokenisation, la vectorisation, la classification et la vérification des résultats. La vectorisation est le facteur que l'on va faire varier. Le reste sera commun à tout les notebooks.

### 2.3.2.1. Nettoyage et tokenisation des commentaires

L'outil `Tokenizer` développé dans keras permet à la fois de filtrer le texte mais également le segmenter en tokens.

Différents paramètres influent sur la séquence qui sera obtenue:

- `num_words`, qui définit le nombre de mots de vocabulaire à considérer. Ce paramètre se base sur la fréquence d'apparition des mots dans le jeu d'entraînement (calculée après usage de la méthode `fit_on_texts`). Il garde les mots les plus fréquents. Ici les petits mots très fréquents n'ont pas été retirés mais en comparaison avec la partie précédente, on considère 20000 mots de vocabulaires contre seulement 5000 précédemment.
- `filters`, qui permet un premier nettoyage du texte. Par défaut, il retire toute la ponctuation, les tabulations et les sauts de ligne. Il fait ceci dit exception pour l'apostrophe ' qui sera conservée.
- `lower`, qui met le texte en minuscule. La case du texte n'a donc plus d'influence.
- `split`, qui sépare les tokens. Par défaut, le séparateur est l'espace.

Une fois le tokenizer appliqué, on obtient une séquence sous forme de liste d'index de mots, où un index correspond à un mot dans la liste des 20000 mots de vocabulaire conservés. On aura donc des indexes entre 1 et 20000.

Pour finaliser les séquences, une dernière étape est nécessaire. Les commentaires ne font pas tous la même longueur. Il faut donc fixer une longueur de séquence (quelques centaines en règle générale, ici on a choisit 100 mots). Pour cela, on utilise l'outil `pad_sequences` développé dans keras. Par défaut des 0 sont ajoutés au début de la séquence pour qu'elle atteigne la longueur voulue (ici 100).

### 2.3.2.2. Vectorisation

Les différents modèles vecteur présentés précédemment ont été testés. Nous avons créé nos propres modèles à partir du vocabulaire de notre jeu de données mais également des modèles déjà entraînés avec un large vocabulaire disponibles en ligne. On parle alors d'apprentissage par transfert (ou "transfer learning" en anglais<sup>14</sup>). Ces modèles permettent généralement une meilleure généralisation du problème de classification traité. La liste des notebooks est la suivante:

<sup>13</sup><https://www.kaggle.com/jhoward/improved-lstm-baseline-glove-dropout/notebook>

<sup>14</sup>[https://en.wikipedia.org/wiki/Transfer\\_learning](https://en.wikipedia.org/wiki/Transfer_learning)

- `google_word2vec.ipynb`, qui utilise le modèle Word2vec pré-entraîné mis à disposition par Google<sup>15</sup>. Ce modèle fait 1.5GB et contient les vecteurs de 3 millions de mots et phrases. Il a été entraîné sur environ 100 milliards de mots provenant d'un jeu de données Google News. La longueur des vecteurs est fixée à 300. Certains stop words ont été retirés ("a", "and" ou "of"). Les erreurs d'orthographe ou de frappe sont présentes (e.g "mispelled" et "misspelled" seront considérés comme deux termes différents). En revanche, un traitement a été effectué pour les mots qui vont par paires. Les espaces ont été remplacés par des \_ (e.g "Soviet\_Union" ou "New\_York")
- `homemade_word2vec.ipynb`, qui utilise notre modèle Word2vec "maison" basé sur le vocabulaire du jeu de données Kaggle. Il contient des vecteurs de dimension 300.
- `pretrained_glove.ipynb`, qui utilise le modèle Glove pré-entraîné disponible sur le site <https://nlp.stanford.edu/projects/glove/>. Celui-ci a été entraîné sur le vocabulaire de Wikipedia 2014 et la cinquième édition de Giga-word<sup>16</sup>. Plusieurs tailles de vecteurs sont disponibles pour le même vocabulaire. On a choisi d'utiliser le modèle de dimension 300 pour le comparer au modèle word2vec.
- `homemade_glove.ipynb`, qui utilise notre modèle GloVe "maison" basé sur le vocabulaire du jeu de données Kaggle. Il contient des vecteurs de dimension 300.
- `pretrained_fasttext.ipynb`, qui utilise le modèle FastText pré-entraîné disponible sur le site <https://fasttext.cc/docs/en/english-vectors.html>. On choisit celui entraîné sur le vocabulaire de Wikipedia 2017, le corpus UMBC<sup>17</sup> et statmt.org. Il contient 1 million de vecteurs de dimension 300.
- `homemade_fasttext.ipynb`, qui utilise notre modèle fasttext "maison" basé sur le vocabulaire du jeu de données Kaggle. Il contient des vecteurs de dimension 300.

### 2.3.2.3. Architecture du réseau de neurones

Comme indiqué précédemment, la structure du réseau est extraite du notebook Kaggle de Jeremy Howard<sup>18</sup>.

```

: 1 inp = Input(shape=(maxlen,))
2 x = Embedding(max_features, embed_size, weights=[embedding_matrix])(inp)
3 x = Bidirectional(LSTM(50, return_sequences=True, dropout=0.1, recurrent_dropout=0.1))(x)
4 x = GlobalMaxPool1D()(x)
5 x = Dense(50, activation="relu")(x)
6 x = Dropout(0.1)(x)
7 x = Dense(6, activation="sigmoid")(x)
8 model = Model(inputs=inp, outputs=x)
9 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

```

Figure 2.7: Architecture du réseau de neurones utilisé

Etudions ensemble cette structure en détails. Dans la figure 2.7, on peut voir que le réseau est constitué de nombreuses étapes:

- l'Input  
Le réseau attend en entrée des vecteurs de dimension 100. Nous avons déjà mis les données sous forme de séquences de longueur 100.
- l'Embedding  
La couche d'embedding convertit les indexes de la séquence `inp` en vecteurs denses de taille fixe. Par exemple les indexes de la séquence `[[4], [20]]` peuvent être transformé en vecteurs

<sup>15</sup><http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/>

<sup>16</sup><https://catalog.ldc.upenn.edu/LDC2011T07>

<sup>17</sup><https://ebiquity.umbc.edu/blogger/2013/05/01/umbc-webbase-corpus-of-3b-english-words/>

<sup>18</sup><https://www.kaggle.com/jhoward/improved-lstm-baseline-glove-dropout/notebook>

de dimension 2  $[[0.25, 0.1], [0.6, -0.2]]$ . Cette couche peut uniquement être utilisée comme première couche d'un modèle. La taille du vocabulaire à considérer est définie par le paramètre `input_dim` (ici `max_features=20000`) et la taille des vecteurs denses définie par le paramètre `output_dim` (ici `embed_size=300`).

Les vecteurs denses sont ceux définis par le modèle Word2vec pré-entraîné et sont précisés en utilisant le paramètre `weights`<sup>19</sup>. Dans ce modèle, on avait 3 millions de vecteurs. On a donc restreint cet ensemble à 20000 vecteurs correspondant aux mots les plus fréquents dans notre jeu d'entraînement. Ce sous-ensemble est appelé matrice d'embedding. Sa dimension est de 20000 lignes et 300 colonnes. En imaginant qu'un mot de notre jeu de données ne soit pas présent dans le modèle Google, il sera alors représenté par un vecteur nul de dimension 300.

Pour résumé, dans notre cas, l'embedding prend en entrée une séquence de longueur 100 et rend en sortie une matrice de dimension (100, 300). Attention! Bien souvent, l'apprentissage d'un modèle se fait par échantillons de taille fixe. On parle de `batch size` en anglais. Au lieu d'entrer une seule séquence dans le réseau, on entre un ensemble de séquences. L'entrée augmente d'une dimension (batch size, longueur de la séquence = 100) ainsi que la sortie (batch size, 100, 300).

- une couche `LSTM` qu'on transforme en réseau `bidirectionnel`  
Une couche de 50 unités LSTM est ajoutée. Elle utilise par défaut la tangente hyperbolique comme fonction d'activation et la sigmoïde pour l'étape de récurrence. Avec le paramètre `recurrent_dropout`, 10% des unités efface leur mémoire lors de l'étape de récurrence.
- le `Pooling`  
L'outil `GlobalMaxPool1D` développé dans Keras permet de réduire la dimension de données temporelles. En effet, le réseau va étudier les caractéristiques des données qui sont parfois en nombre important. Pour réduire les temps de calcul, on réduit souvent la dimension des données avec une étape de pooling. Ici les données d'entrée sont de dimension 3 de la forme (batch size, 100, 100) et la sortie en dimension 2 de la forme (batch size, 100) (voir la figure 2.8).
- une couche `Dense`  
Une fois la dimension des données réduite, on applique une couche standard de neurones. Les neurones de cette couche fournissent à la fonction d'activation le produit des données d'entrée et une matrice de poids créée par la couche, auquel un biais est ajouté. Ici, la couche contient 50 neurones. La sortie sera donc de dimension (batch size, 50). La fonction d'activation est la `relu`<sup>20</sup> qui correspond à  $f(x) = x^+ = \max(0, x)$ .
- un `Dropout`  
On applique un dropout à la sortie de la couche dense pour remettre à 0 la valeur de 10% des unités de la couche durant chaque étape de l'entraînement. Cela permet d'éviter le sur-apprentissage[5].
- une dernière couche `Dense`  
Cette couche de 6 unités permettra la classification des commentaires. Sa fonction d'activation est la sigmoïde  $f(x) = \frac{1}{1 + e^{-x}}$ .

Une fois l'architecture du réseau définie, on paramètre le modèle pour l'entraînement avec la fonction `compile`. Il faut définir une fonction d'optimisation qui permet en général de réduire le temps de calcul lors de l'entraînement du modèle. Ici, on choisit la fonction `adam`<sup>21</sup>, une extension de la descente de gradient stochastique. De même, on doit définir une fonction de loss. Dans le cas de notre problème multi-label avec une fonction d'activation sigmoïde, on utilise la `binary cross entropy loss`<sup>22</sup>. Contrairement à la fonction `softmax`, la `binary cross entropy` (BCE) est indépendante pour chaque classe. C'est pourquoi elle est recommandée pour la classification multi-label. La décision

<sup>19</sup>[https://github.com/keras-team/keras/blob/master/keras/engine/base\\_layer.py](https://github.com/keras-team/keras/blob/master/keras/engine/base_layer.py)

<sup>20</sup>Rectified Linear Unit

<sup>21</sup><https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

<sup>22</sup>[https://gombru.github.io/2018/05/23/cross\\_entropy\\_loss/](https://gombru.github.io/2018/05/23/cross_entropy_loss/)

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 100)	0
embedding_1 (Embedding)	(None, 100, 300)	6000000
bidirectional_1 (Bidirection	(None, 100, 100)	140400
global_max_pooling1d_1 (Glob	(None, 100)	0
dense_1 (Dense)	(None, 50)	5050
dropout_1 (Dropout)	(None, 50)	0
dense_2 (Dense)	(None, 6)	306
Total params: 6,145,756		
Trainable params: 6,145,756		
Non-trainable params: 0		

Figure 2.8: Dimension des données au cours des différentes étapes de notre modèle

prise pour une classe n'impacte pas la décision des autres classes. Pour chaque classe, on a une décision binaire: le commentaire appartient à cette classe ou non. Pour calculer la loss globale, on somme la loss de chaque problème binaire (6 problèmes dans notre cas).

$$BCE = \begin{cases} -\log(f(s_1)) & \text{if } t_1 = 1 \\ -\log(1 - f(s_1)) & \text{if } t_1 = 0 \end{cases}$$

Si  $t_1 = 1$ , le commentaire appartient à la classe  $C_1$ .  $s_1$  représente le score de la sigmoïde. La fonction de loss `hinge` est également intéressante dans le cas d'une classification multi-label combinée à la fonction d'activation tangente hyperbolique mais il faudra transformer les labels en  $-1$ ,<sup>23</sup>.

La fonction d'activation `tangente hyperbolique` s'écrit  $f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$  et la fonction de loss `hinge`  $\ell(y) = \max(0, 1 - t \cdot y)$  où  $t$  est la classe attendue ( $\pm 1$ ) et  $y$  le score brut du classifieur. Quand  $t$  et  $y$  ont le même signe et  $|y| > 1$ ,  $y$  prédit la bonne classe ( $\ell(y) = 0$ ).

On choisit par défaut la mesure d'accuracy dans la définition du modèle mais celle-ci donne une moyenne des accuracys pour chaque label. On veut des mesures plus précises. On réutilise l'accuracy et la matrice de confusion de `sklearn` utilisés précédemment. Pour le calcul du recall, de la précision et de la F-mesure, on utilise la fonction `classification_report`.

On peut maintenant visualiser et interpreter les résultats!

<sup>23</sup><https://github.com/keras-team/keras/issues/10371>



# 3

## Résultats

### 3.1. Comparaison des modèles de représentation vectorielle

Après avoir utilisé trois outils de représentation vectorielle du texte, Word2vec, GloVe et fasttext, il est maintenant temps de comparer leur performance à modèle d'apprentissage égal. On a également étudié des méthodes de transfer learning en comparaison à des représentations basées sur le vocabulaire spécifique de notre jeu de données.

Comme évoqué à la fin de la partie 2.3.2.3, on peut mesurer de différente façon l'accuracy d'un modèle.

L'accuracy proposée par Keras représente la moyenne des accuracy 1 contre tous (1vsAll) tandis que celle proposée par Sklearn mesure la proportion de classes toutes bien prédites en même temps dans le cas du problème multilabel.

Accuracy	Transfer learning			Homemade		
	Word2vec	GloVe	FastText	Word2vec	GloVe	FastText
Keras	0.972907	0.974431	nan	0.970513	0.969812	0.973269
Sklearn	0.885382	0.891884	nan	0.873612	0.874925	0.886992
Sklearn 1vsAll						
Toxic	0.927584	0.933211	nan	0.917768	0.918128	0.929835
Severe Toxic	0.993591	0.992669	nan	0.993435	0.994013	0.994248
Obscene	0.963127	0.965144	nan	0.962455	0.955609	0.962471
Threat	0.996701	0.996780	nan	0.996670	0.996514	0.996639
Insult	0.9650192	0.966785	nan	0.961330	0.963034	0.965394
Identity Hate	0.991418	0.991997	nan	0.991418	0.991575	0.991028

Figure 3.1: Tableau des résultats de classification obtenus avec différentes méthodes de représentation vectorielle

Via la figure 3.1, on peut relever plusieurs informations:

- On constate que globalement les résultats de classification sont meilleures après avoir utilisé le modèle pré-entraîné GloVe. La co-occurrence des mots semble donc être une information pertinente pour notre jeu de données.
- A méthode égale, on constate que le transfer learning est toujours légèrement plus performant qu'un modèle maison. En effet, les modèles pré-entraînés sont entraînés sur un très large jeu de données, ce qui assure une représentation vectorielle des mots plus fiable. Plus le jeu d'entraînement est grand, plus la représentation des mots qui le compose est fidèle à la réalité. Le transfer learning contient, de plus, un large vocabulaire qui permet une utilisation générale du modèle. A l'opposé notre modèle maison risque d'être trop spécifique aux textes à caractère injurieux. Il serait intéressant d'appliquer, par exemple, notre modèle GloVe maison à un autre

problème concernant du texte non injurieux (e.g poésie, texte de loi, etc.) pour voir sa capacité à généraliser. On peut imaginer que le vocabulaire de notre modèle sera trop restreint pour couvrir des domaines plus généralistes.

- On ne peut pas tirer de conclusions sur le modèle pré-entraîné de FastText, car nous avons expérimenté des problèmes de disparition de gradient qui restent à ce jour encore inexpliqués. Le modèle nous renvoie alors des prédictions nulles.
- Lorsque l'on souhaite mesurer le nombre de classe toutes correctement prédites pour un commentaire, l'accuracy diminue fortement (Sklearn). Il est plus facile de prédire les classes correctement une à une plutôt que de les prédire toutes correctement en même temps.
- De manière générale chaque classe semble être bien prédite indépendamment des autres (1vsAll). La prédiction de notre modèle est toujours correcte au delà de 91%. Mais attention! La distribution non uniforme des données joue beaucoup dans l'interprétation des résultats du modèle. C'est pour quoi il est important de mesurer le recall, précision et f-mesure.

Pour la suite, on considère les résultats du modèle GloVe pré-entraîné. Ici la classification de Threat est de 99.68% mais lorsqu'on observe la matrice de confusion de la classe threat contre les autres, on réalise vite que threat n'est que rarement prédite correctement. En effet dans la figure 3.4, on peut voir que la classe threat est très peu représentée dans le jeu de données (seulement 211 commentaires du jeu test sont labélisés threat sur 63978). Le modèle prédit donc plus souvent un commentaire threat

```
... Traitement du label threat
L'accuracy du jeu test est 0.9967801431742161
Matrice de confusion sur le jeu test:
[[63760    7]
 [  199   12]]
Rapport :
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	63767
1	0.63	0.06	0.10	211

Figure 3.2: Matrice de confusion de la classe threat (1) contre les autres (0)

dans la classe "autres" (faux négatifs: 199) que dans la classe "threat" (vrai positifs:12) d'où un recall très faible (6%). A l'inverse, il ne classe quasiment jamais les commentaires autres dans la classe "threat" (faux positifs: 7) d'où une précision assez faible également pour la classe threat (63%)

## 3.2. Discussion

```

... Traitement du label identity_hate
L'accuracy du jeu test est 0.9919972490543625
Matrice de confusion sur le jeu test:
[[63078  188]
 [  324  388]]
Rapport :

```

	precision	recall	f1-score	support
0	0.99	1.00	1.00	63266
1	0.67	0.54	0.60	712

Figure 3.3: Matrice de confusion de la classe identity\_hate (1) contre les autres (0)

```

... Traitement du label toxic
L'accuracy du jeu test est 0.9332114164243959
Matrice de confusion sur le jeu test:
[[54891  2997]
 [ 1276  4814]]
Rapport :

```

	precision	recall	f1-score	support
0	0.98	0.95	0.96	57888
1	0.62	0.79	0.69	6090

Figure 3.4: Matrice de confusion de la classe toxic (1) contre les autres (0)



4

## Conclusion

-> test modèle sur d'autres jeux de données?



# Liste d'abréviations

BCE : Binary Cross Entropy

BRNN : Bidirectionnal Recurrent Neural Network

CBOW : Continuous Bag-Of-Words

HS : Hierarchical Softmax

LSTM : Long Short Term Memory

NLP : Natural Language Processing

NS : Negative Sampling

RELU : Rectified Linear Unit

RNN : Recurrent Neural Network





# Bibliography

- [1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.
- [2] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.
- [3] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. pages 1532–1543, 2014.
- [4] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.