# Transparence des algorithmes de Deep Learning pour les données textuelles assurantielles

C. Francon, K. Larbi et W. Sanchez

15 février 2021

## Problématique

La Société Générale Assurances collecte et reçoit une grosse volumétrie de données textuelles à travers différents canaux et notamment par mail. Ces mails proviennent d'agents du réseau ou de clients et sont centralisés par un département (par exemple, lorsqu'un client souhaite se renseigner, gérer ses contrats ou se plaindre auprès d'un service). Ces messages doivent être redirigés vers le bon département : initialement, un opérateur humain lisait tous ces mails et triait en fonction du contenu vers le service le plus pertinent. Compte tenu du coût pour réaliser cette action, des modèles de classification textuelle ont été développés au sein du Datalab de la Société Générale Assurances afin d'assigner à chaque mail reçu, un service vers lequel il devrait être transféré.

Depuis quelques années, les utilisations des réseaux de neurones se multiplient. En effet, ces modèles permettent de répondre à un large spectre de problématiques : classification d'images en utilisant des réseaux de neurones à convolution (ou *Convolutional Neural Network*) (AlexNet [2012]), traitement du langage par l'intermédiaire de réseaux de neurones (Word2Vec [2013], FastText [2016], Bert [2018]).

Malgré des performances remarquables, ces méthodes souffrent d'un manque d'interprétabilité. Il est délicat d'obtenir des éléments expliquant les prédictions d'un modèle constituant généralement un frein pour les équipes métiers. Cette problématique est connue en *machine learning*.

Le but de ce projet est d'utiliser des méthodes de transparence des algorithmes sur une sélection de modèles et de comparer les résultats à des données réelles.

## Présentation des données

Le jeu *allocine_review* est un jeu de données contenant des avis sur des films issus du site allocine. Les avis ont été écrits entre 2006 et 2020. Pour chaque avis, l'utilisateur indique une note comprise entre 0.5 et 5 sur 5 : si la note est inférieure ou égale à 2 alors l'avis est considéré comme négatif sinon il est considéré comme positif.

Ce jeu de données, disponible en libre accès, permet de tester les différents modèles avant l'utilisation de données issues de la Société Générale.

Le jeu est découpé en trois sous-jeux : le jeu d'entrainement (160 000 avis), le jeu de validation (20 000 avis) et le jeu de test (20 000 avis).

Les avis subissent des prétraitements (*preprocessing*). Ces traitements permettent, par exemple, de découper une phrase en une liste de mots (ou même caractères) ou encore de prendre la racine d'un mot pour éviter de coder de manière différente des mots très proches (par exemple : "chantez", "chant", "chanson" renvoient à la même idée, ces trois mots seront remplacés dans les phrases par le mot "chant"). Certains mots, appelés *stop-word*, apportent peu d'informations pour la prédiction (par exemple, les prépositions) : ils sont donc supprimés lors des traitements de la base de données.

Pour appliquer les méthodes présentées ultérieurement, il faut que chaque avis ait le même nombre de mots. L'illustration suivante présente la distribution du nombre de mots. En prenant $s = 67$, seuls 20% des avis seront tronqués.

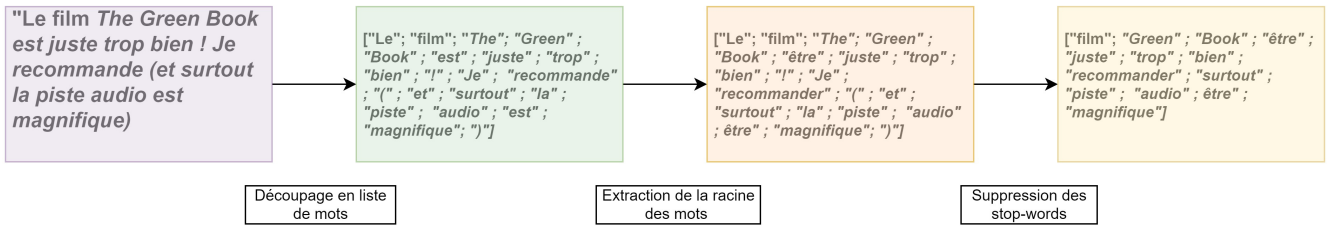


Figure 1: Principe de la *tokenisation*

# Présentation des modèles

### Embedding

Le but des modèles présentés ci-après est de proposer des modèles qui, à partir d'une phrase, permettent de prédire une variable binaire d'intérêt (par exemple, "l'avis est positif"/"l'avis est négatif"). La variable binaire est donc décrite par une variable prenant comme valeur soit 0 soit 1.

Néanmoins, il est impossible d'utiliser directement les phrases dans un modèle : il faut les transformer en vecteurs. L'application qui à chaque mot associe un vecteur est appelée plongement lexical (ou *embedding*). Il existe plusieurs types d'*embedding* dans la littérature, seuls trois seront considérés dans cette note :

- One Hot Encoding : le vocabulaire $\mathcal{V}$ est défini comme l'ensemble des mots apparaissant dans les documents. Pour chaque mot $w$ appartenant au vocabulaire, un indice $i$ est associé. L'*embedding One Hot Encoding* de $w$ noté $ohe(w)$ est donné par le vecteur de $\underbrace{(0, ..., 1, 0, ...0)}_{\text{le 1 étant à la ième composante}} \in \{0,1\}^{\text{Card}\mathcal{V}}$. Card$\mathcal{V}$ sera appelé dimension de l'*embedding*. Cette dimension est ici dépendante des données .
  Le *One Hot Encoding* présente deux défauts majeurs :
  - les vecteurs appartiennent à un espace de grande dimension et sont creux (Card($\mathcal{V}$) >10 000 ici et pour chaque vecteur, seul une composante est non nulle)
  - certaines opérations arithmétiques comme l'addition n'ont pas de sens mathématique ou linguistique. Par exemple, la somme de deux vecteurs en *One Hot Encoding* ne correspond pas à un vecteur *One Hot Encoding*.
- Word2Vec [Mikolov, 2012] : les vecteurs sont décrits comme des vecteurs de $\mathbb{R}^D$ où $D$ est un entier appelé dimension de l'*embedding* (la construction de ces vecteurs ne sera pas décrite ici). Cette dimension n'est pas dépendante des données. De plus, l'addition a un sens linguistique : w2v("Paris") $\approx$ w2v("Rome") - w2v("Italie") + w2v("France") où w2v($w$) décrit le vecteur Word2vec associé au mot w. Un inconvénient du modèle Word2Vec est qu'il n'est pas possible de fournir une représentation vectorielle.
- FastText [Bojanowski, 2016] : les mots sont découpés en sous-mots d'une taille donnée et chaque sous-mot est transformé en vecteur à l'aide d'un *embedding* puis ces vecteurs sont agrégés (en faisant la moyenne généralement). Un avantage du FastText par rapport au modèle Word2Vec est que cet *embedding* est moins sensible aux fautes d'orthographes et permet également de fournir une représentation vectorielle d'un mot jamais rencontré.

## Modèles

Différents modèles sont proposés afin d'essayer de prédire si un avis est positif ou négatif à l'aide du contenu de ce dernier. Un premier modèle basé sur une stratégie *Bag of Words* et d'un modèle de régression logistique est développé afin de servir de référence pour la comparaison des modèles (il s'agira du *benchmark*).

Les modèles suivants se basent sur deux types d'architecture d'apprentissage profond utilisés à la Société Générale (ou *Deep Learning*) : les réseaux de neurones à convolution (CNN) et les réseaux de neurones récurrents (*RNN*).

Les réseaux de neurones à convolution permettent d'obtenir de nouvelles caractéristiques en utilisant des filtres sur les données alors que les réseaux de neurones récurrents utilisent des cellules cachées permettant de garder une mémoire plus longue lors de l'apprentissage.

Le modèle testé initialement se base sur un *embedding* Word2Vec afin d'obtenir une représentation numérique des mots ainsi que d'un CNN. Ce modèle est sensible aux fautes d'orthographe : une solution est de ne plus travailler à une maille "mot" mais à une maille "caractère".

L'intérêt de l'ajout du bi-LSTM est qu'il permet de garder une mémoire à long-terme. Ainsi, il permet de mettre en lien des mots possiblement très éloignés dans la phrase.

Ces modèles seront à la base de nos travaux sur l'interprétabilité car ce sont les architectures les plus utilisées en classification.

### Modèle basé sur une régression logistique et une stratégie *Bag of Words*

Dans un premier temps, il a été développé un modèle de classification textuelle "simple".

Ainsi, c'est un modèle de régression logistique basé sur des sacs de mots (BoW) qui a été développé. Pour ce faire, les $n$ mots les plus fréquents du jeu d'entraînement sont relevés et constituent le dictionnaire. La variable à expliquer est binaire (avis positif ou négatif) et les covariables correspondent à la fréquence d'apparition de chaque mot du dictionnaire au sein du commentaire Allocine (ainsi chaque avis est décrit par un vecteur de $n$ composantes).

Ici, la taille du dictionnaire est limitée à 100 mots en raison de problème de convergence et de capacité mémoire.

L'AUC (*Area Under the Curve*) peut s'interpréter comme la probabilité que le modèle assigne une probabilité plus forte aux commentaires positifs que négatifs. Ainsi, si la valeur de l'AUC est de 0.5, cela signifie que le modèle a d'aussi bons résultats que le hasard. De plus, le taux de bien classé (la précision) est de **82%**.

L'entraînement du modèle a nécessité **7 minutes** et permet d'obtenir une AUC (Area Under the Curve) de **0.74**.

Ces métriques vont permettre de comparer la performance des architectures entre elles.

### Modèle Embedding suivi d'un réseau de neurones convolutionnels

L'article de Zhang *et al* [3] propose une analyse de sensibilité des réseaux de neurones convolutionnels pour la classification binaire de phrases. Les réseaux de neurones convolutionnels sont des modèles permettant d'utiliser la dépendance spatiale entre les variables explicatives. Dans notre cas, le CNN permet donc de prédire une variable d'intérêt en considérant à chaque fois $f$ mots consécutifs.

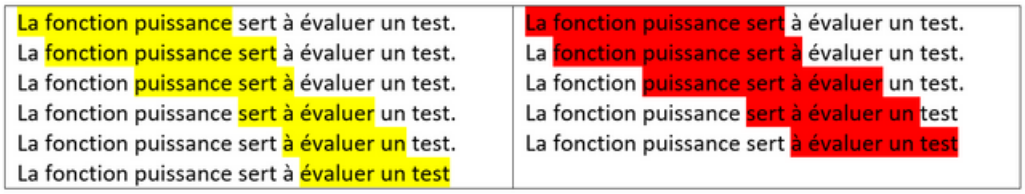


Figure 2: Ici deux types de filtres : des filtres de taille 3 et des filtres de taille 4 - l'intégralité de la phrase est ainsi parcourue

Le réseau décrit [3] correspond à un réseau avec trois couches cachées :

- la première permet de transformer chaque mot en vecteur. Cette couche agit comme un *embedding*;
- la deuxième permet de passer des filtres sur les différentes phrases afin d'extraire des caractéristiques. Il s'agit d'une couche convolutionnelle constituée de filtres unidimensionnels.
- la dernière permet d'aider à mieux prédire les probabilités d'appartenance à une classe.

Les auteurs proposent une méthode pour définir les valeurs optimales de certains paramètres : nombre de mots consécutifs considérés par le réseau pour prédire, nombre de caractéristiques créées à partir de ces filtres, régularisation pour éviter le sur-apprentissage ...

**Modèle CNN à la maille caractère**

Contrairement à la modélisation précédente [3] qui utilise des vecteurs de mots pour classifier les commentaires *allocine*, cette fois-ci sont considérés des vecteurs de caractères [4]. En considérant la maille caractère, **l'idée est de pouvoir s'affranchir des contraintes liées aux fautes d'orthographes et aux mots inconnus**.

Par exemple, dans le jeu de données *allocine_review*, à la place d'entraîner le modèle avec une succession de mots composant le commentaire, une succession de caractères sert d'intrant. Chaque caractère est alors représenté par un vecteur *One Hot Encoding* (comme décrit précédemment mais à une maille caractère), de dimension égale à l'alphabet. L'alphabet considéré ici est composé de toutes les lettres de l'alphabet latin (minuscules et majuscules) et de caractères spéciaux (exemple : , ! é à). En limitant à 1014 caractères la taille d'un commentaire, ce dernier est donc représenté par une matrice [1014x83].

Comme cité précédemment, cette méthode a comme avantage que les fautes d'orthographe, les émoticônes ou bien les combinaisons anormales de caractères peuvent être considérées et apprises par le modèle.

Cependant, cette architecture a certains inconvénients dont un principal : pour que cette approche soit plus intéressante que d'autres méthodes plus simple, elle nécessite énormément de données (des millions d'observations [4]). Cependant, le jeu de données ne contient pas des millions de commentaires.

**Modèle Bi-LSTM et CNN**

Le troisième modèle que nous avons implémenté est celui décrit par l'article [5]. Ce modèle est caractérisé par trois couches principales.

Tout d'abord, une couche d'*embedding* qui prend en entrée un vecteur comprenant les indices des mots dans le vocabulaire et qui associe à chaque mot son vecteur d'*embedding*. Les poids de cette couche sont initialisés avec un *embedding* word2vec pré-entrainé de dimension 200. Nous avons également fixé la taille des phrases à 67 en retirant des mots si la phrase est trop longue et rajoutant des 0 la phrase est trop courte (comme dans le premier modèle [3]). En sortie de cette couche nous avons donc une matrice de taille [200x67].

La deuxième couche est une couche bi-LSTM (ou *Bidirectional Long Short-Term Memory*). Il s'agit d'une architecture de RNN (*recurrent neural network*). L'idée d'un RNN classique est de garder une trace à chaque étape d'apprentissage et de prendre en compte le contexte passé. LSTM a été créé pour résoudre le problème de « disparition » ou d' « explosion » du gradient (*Vanishing and exploding gradient problem*). En effet, lors de la rétropropagation, les gradients de la fonction de coût sont calculés par la règle de la chaîne. Ainsi, on multiplie successivement ces gradients et s'ils sont petits au départ, on se retrouve avec des gradients très faible en début de chaîne et donc les premiers poids ne se mettent pas à jour. À l'inverse, le gradient explose si les gradients sont grands au départ. Une couche bi-LSTM permet de prendre en compte le contexte de gauche (mots précédents) et le contexte de droite (mots suivants).

La troisième couche est une couche de convolution. Enfin, ce modèle se termine par une couche *max-pooling* puis *fully-connected*.

**Récapitulatif des résultats**

| Modèles | AUC | Précision | Temps de calcul | Matériel utilisé |
|---|---|---|---|---|
| Bag of Words (n=100) | 0.74 | 82% | 7 min | CPU |
| Embedding (Fast Text) + CNN | 0.97 | 91% | 6 min | Google Colab |
| Caractères CNN | 0.97 | 92% | 37 min | GPU |
| Embedding (w2v) + Bi-LSTM + CNN | 0.98 | 93% | 20 min | GPU |

Table 1: Résultats sur le jeu de test

# Limites et difficultés rencontrées

Les premières difficultés auxquelles nous avons fait face ont été les différents coups d'entrée pour comprendre, s'approprier et mettre en application ces papiers.

En effet, dans un premier temps, il nous a fallu prendre en mains les articles de recherche sur ces architectures ; la finalité était de les implémenter sur Python à l'aide de PyTorch. Cela nous a donc demandé un certain temps pour nous documenter, notamment à propos des CNN, des fonctions d'attention ou bien des LSTM.

De plus, nous ne connaissions pas PyTorch, le *framework* de *Deep Learning* développé par Facebook. Cela a été une contrainte supplémentaire de comprendre comment intéragir avec ces différents objets. Néanmoins, c'était probablement la contrainte la plus simple à dépasser car beaucoup de ressources traitant de PyTorch sont disponibles sur internet.

La dernière difficulté a été d'utiliser Kedro, un *framework* de développement de projet en *data science*. Le principe de Kedro est de pouvoir contenir au sein de pipeline les différentes fonctions de transformations des données et d'entrainement des modèles. L'intérêt est de pouvoir reproduire, maintenir et travailler plus facilement tout ce qui a trait aux modèles. C'est le *framework* de développement de projet *data science* qui est utilisé au sein de la Société Générale Assurances.

Enfin, la grande limite à laquelle nous avons fait face est notre capacité computationnelle. En effet, toutes ces architectures sont gourmandes en ressources. Alors, plusieurs solutions ont été trouvées pour résoudre partiellement ce problème : utiliser Google Colab ou bien CUDA. Néanmoins, le jeu de données à notre disposition est relativement peu volumineux et cela contribue à accélérer l'entraînement des modèles.

## Perspectives

Le but initial du projet est de s'intéresser à la transparence des modèles. Nous avons notamment en vue une méthode particulière : grad-CAM (*Gradient-weighted Class Activation Mapping*). Il s'agit d'une méthode utilisée dans le cadre du traitement d'image pour un modèle CNN et elle a encore été très peu utilisée dans une problématique NLP (*Natural Language Processing*). Elle permet de visualiser les régions de l'image d'entrée qui ont été importantes pour les prédictions du modèle. On obtient une carte d'activation pour chaque classe et chaque couche de convolution. Nous essaierons d'utiliser cette méthode pour mettre en lumière quels mots dans la phrase ont eu le plus de poids lors de la prédiction.

Enfin, nous utiliserons comme benchmark les sur-modèles explicatifs suivants : LIME, Shap et Deeplift.

## Conclusion

L'implémentation des modèles, qui constitue la première étape du projet, est terminée. Il ne reste qu'à ajuster les hyperparamètres de nos modèles, ce qui sera facilité par Kedro et MLflow.

Toutefois, il est nécessaire de se rappeler que le but du projet est de développer des outils pour interpréter la sortie des modèles. Or, nous n'avons pas encore commencé à implémenter ces méthodes. Aussi, nous avons amorcé le travail sur les articles traitant du grad-CAM, du LIME etc.

## Bibliographie

[1] Deep Learning, *Ian Goodfellow and Yoshua Bengio and Aaron Courville*

[2] Dive into Deep Learning, https://d2l.ai/

[3] *Ye Zhang and Byron C. Wallace*, A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification, Proceedings of the The 8th International Joint Conference on Natural Language Processing, pages 253-263, 2017.

[4] *Xiang Zhang, Junbo Zhao and Yann LeCun*, Character-level Convolutional Networks for Text Classification, Advances in Neural Information Processing Systems 28, pages 649-657, 2015.

[5] *Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao and Bo Xu*, Text Classification Improved by Integrating Bidirectional LSTM
with Two-dimensional Max Pooling, Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 3485-3495, 2016.