

ÉCOLE NATIONALE DE LA STATISTIQUE
ET DE L'ADMINISTRATION ÉCONOMIQUE



Groupe de statistique appliquée

Sujet 68 : Transparence des algorithmes de Deep Learning pour les données textuelles assurantielles

Tuteurs :

HU François
HONORE-ROUGE Yolan

Etudiants :

LARBI Khaled
SANCHEZ Wenceslas
FRANCON Camille

Remerciements

Nous adressons nos remerciements aux personnes qui nous aidé dans la réalisation de ce projet, pour la mise à disposition de ce sujet, leurs conseils, leurs aides et leur suivi tout le long de cette année.

Merci à M. HONORE-ROUGE Yolan, Actuaire et Data Scientist à la Société Générale Assurance, et à M. HU François, Machine Learning Engineer and Researcher à la Société Générale Assurance et Doctorant en mathématiques appliquée. Leurs explications et leurs conseils quant à l'utilisation des différents *framework* du projet nous ont beaucoup aidé pour la conception et l'élaboration de nos programmes.

Leur pédagogie et leurs éclaircissements nous ont été d'une grande aide pour mieux comprendre la problématique, les articles de recherche et ainsi mieux répondre aux problèmes soulevés par ce sujet.

Sommaire

Table des figures	3
Introduction	4
1 Présentation des données et des modèles	5
1.1 Données	5
1.2 Modèles	5
1.3 Récapitulatifs des résultats	8
1.4 Environnement de travail	8
2 Vers l'interprétabilité	9
2.1 Présentations des méthodologies	9
2.1.1 Exploiter l'architecture CNN des modèles : GradCAM	9
2.1.2 Méthodes d'interprétabilité agnostique : LIME et SHAP	12
2.2 Comparaison des résultats	15
2.2.1 Résultats du GradCAM au global	15
2.2.2 Exemples atypiques	18
3 Limites et améliorations	27
3.1 Limites computationnelles et logistiques	27
3.2 Limites inhérentes aux modèles	27
Conclusion	28
Annexes	29
Bibliographie	37

Table des figures

1	Principe de la <i>tokenisation</i>	5
2	Réseau de neurones et calcul par couche - Source : researchgate.net	6
3	Ici deux types de filtres : des filtres de taille 3 et des filtres de taille 4 - l'intégralité de la phrase est ainsi parcourue	7
4	Features maps : issue de https://www.semiconductorforu.com/artificial-intelligence-impacts-automotive-design/a-cnn-breaks-an-image-into-feature-maps/	9
5	Architecture des modèles acceptés par CAM - Source : [Sel+17]	10
6	Construction des Class Activation Map - Source : [Zho+15]	10
7	Exemple de méthode d'upsampling - Source : cs231n.stanford.edu	11
8	Illustration de LIME pour de la classification d'image. Source : Rebeiro (2016)	14
9	Fréquence des mots les plus importants pour la classe 1, pour les commentaires classifiés positifs	16
10	Fréquence des <i>bi-gram</i> les plus importants pour la classe 1, pour les commentaires classifiés positifs	17
11	Application de GradCAM pour la classe 1 sur l'exemple 516	19
12	Application de LIME sur l'exemple 516	19
13	Application de SHAP sur l'exemple 516	20
14	Application de GradCAM pour la classe 1 sur l'exemple 8619	21
15	Application de LIME sur l'exemple 8619	21
16	Application de SHAP sur l'exemple 8619	22
17	Application de GradCAM pour la classe 0 sur l'exemple 1287	23
18	Application de LIME sur l'exemple 1287	23
19	Application de SHAP sur l'exemple 1287	24
20	Application de GradCAM pour la classe 1 sur l'exemple 13342	25
21	Application de LIME sur l'exemple 13342	25
22	Application de shap sur l'exemple 13342	26
23	Matrice de confusion	29
24	Visualisation de l'architecture du projet permise grâce à Kedro	30
25	Fréquence des mots les plus importants pour la classe 0, pour les commentaires classifiés négatifs	33
26	Fréquence des bi-grams les plus importants pour la classe 0, pour les commentaires classifiés négatifs	34
27	Application de GradCAM pour la classe 0 sur l'exemple 516	35
28	Application de GradCAM pour la classe 0 sur l'exemple 8619	35
29	Application de GradCAM pour la classe 1 sur l'exemple 1287	36
30	Application de GradCAM pour la classe 0 sur l'exemple 13342	36

Introduction

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. Actuellement seuls les cas complexes ou particuliers sont traités par cet opérateur. 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 [KSH12]), traitement du langage par l'intermédiaire de réseaux de neurones (Word2Vec [Mik+12], FastText [Boj+16], Bert [Dev+19]).

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 bien connue en *machine learning*, allant même jusqu'à renommer les réseaux de neurones de *modèles boîtes noires*.

Le but de ce projet est donc d'étudier des méthodes de transparence des algorithmes sur une sélection de modèles et de comparer les résultats à des données réelles. On s'intéressera à trois méthodes : une qui exploitera la structure du modèle d'apprentissage (GradCAM) et deux méthodes agnostiques d'interprétabilité (LIME et SHAP). Il est important de noter que contrairement aux deux méthodes agnostiques, il n'existe pas de librairie qui implémente la méthode GradCAM. Il a donc fallu développer une solution pour l'utiliser au sein des modèles qui vont être présentés. C'est ce que vous pourrez retrouver sur la page Github du projet (disponible [ici](#)).

Dans la première partie de ce rapport, le contexte de ce projet sera présenté puis les différents réseaux de neurones, les données et les infrastructures utilisés vont être introduits. Dans une autre partie, après avoir entraîné les différents modèles, les méthodes d'interprétabilité et leurs applications sur certains exemples seront présentées. Des conclusions seront données sur la compatibilité des méthodes d'interprétabilité et les modèles d'apprentissage utilisés. Enfin dans une dernière parties, les limites techniques, méthodologiques et statistiques de ce projet seront proposés ainsi que des pistes d'améliorations.

1 Présentation des données et des modèles

1.1 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'entraînement (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. Ce processus nécessaire afin d'éviter des problèmes de surapprentissage (mot rare par exemple) ou d'augmenter artificiellement la dimensionnalité du problème.

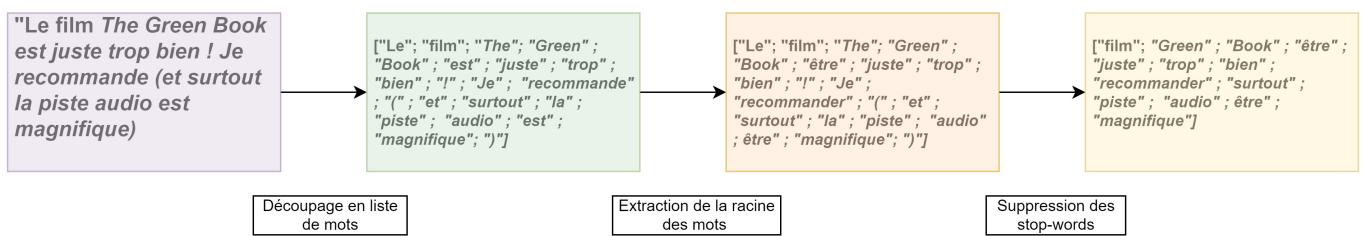


FIGURE 1 – Principe de la *tokenisation*

Pour appliquer les méthodes présentées ultérieurement, il faut que chaque avis ait le même nombre de mots. La figure 1 présente la distribution du nombre de mots. En prenant 67 mots, seuls 20% des avis seront tronqués.

1.2 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'encondage *One Hot Encoding* de w noté $ohe(w)$ est donné par le vecteur de $\underbrace{(0, \dots, 1, 0, \dots 0)}_{\text{le } 1 \text{ étant à la } i\text{ème composante}} \in \{0, 1\}^{\text{Card}\mathcal{V}}$. $\text{Card}\mathcal{V}$ sera appelé dimension de l'*encodage*. Cette dimension est ici dépendante des données .

Une série de mots peut être représentées à l'aide de l'encodage *One Hot Encoding* de chacun des mots : il suffit de considérer la somme des encodages. Cette représentation d'une série de mots (typiquement une phrase) est nommée *Bag-Of-Words*.

Le *One Hot Encoding* présente deux défauts majeurs :

- les vecteurs appartiennent à un espace de grande dimension et sont creux ($\text{Card}(\mathcal{V}) > 10\ 000$ 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 [Mik+12] : 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 de mots non rencontrés pendant l'apprentissage.
- FastText [Boj+16] : 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'orthographies et permet également de fournir une représentation vectorielle d'un mot jamais rencontré.

Les modèles étudiés

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* (BoW) 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 réseaux de neurones sont des méthodes d'apprentissage supervisée pouvant prendre en entrant une grande variété de données. Un réseau de neurones est constitué de couches : dans chaque couche, des calculs sont réalisés (généralement, des sommes pondérées et l'application d'une fonction non-linéaire) puis le résultat est envoyé à la couche suivante comme indiqué sur la figure 2. La dernière couche, appelée *couche de sortie* (ou *output layer*), correspond à la prédiction du réseau. Les poids sont optimisés afin de minimiser une fonction objectif.

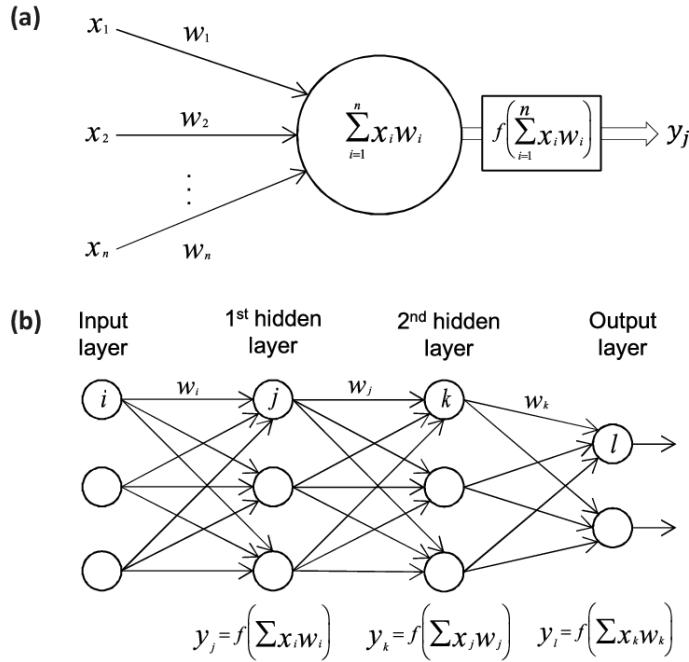


FIGURE 2 – Réseau de neurones et calcul par couche - Source : researchgate.net

Il existe différents types de réseaux de neurones en fonction des tâches à réaliser, des données en entrant ou du type de résultats souhaité. Dans le cadre de ce projet, les réseaux suivants sont considérés :

- Les réseaux de neurones à convolution (CNN) 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".

- Les réseaux de neurones récurrents (RNN) permettant de garder en mémoire certaines informations après chaque étape d'entraînement. Le modèle biLSTM sera en particulier étudié car 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.

a. 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". Il s'agit d'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).

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. L'entraînement du modèle a nécessité **7 minutes** et permet d'obtenir un modèle dont l'AUC (Area Under the Curve) est de **0.74** et le taux de bien classé (précision) est de **82%**. D'autres résultats sont disponibles à la table 1.

Ces métriques vont permettre de comparer la performance des architectures.

b. Modèle Embedding suivi d'un réseau de neurones convolutionnels

L'article de Zhang *et al* [Kim14] 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.

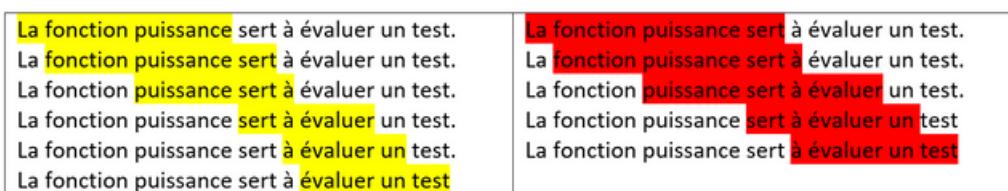


FIGURE 3 – Ici deux types de filtres : des filtres de taille 3 et des filtres de taille 4 - l'intégralité de la phrase est ainsi parcourue

Le réseau décrit [Kim14] 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 ...

c. Modèle CNN à la maille caractère

Contrairement à la modélisation précédente [Kim14] qui utilise des vecteurs de mots pour classifier les commentaires *allocine*, cette fois-ci sont considérés des vecteurs de caractères [ZZL15]. 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*, 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.

Le modèle est composé de six de couches de convolution successives, à l'instar du modèle VGG16 [SZ14] (composé de dix-neuf couches dont seize couches de convolution) puis de trois couches de *fully-connected*. Cette architecture est similaire à ce qui est couramment utilisé pour la classification d'image.

Comme dit 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 simples, elle nécessite énormément de données (des millions d'observations [ZZL15]). Cependant, le jeu de données ne contient pas des millions de commentaires.

d. Modèle Bi-LSTM et CNN

Le troisième modèle implémenté est celui décrit par l'article [Zho+16]. 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 entraîné 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 zéros si la phrase est trop courte (comme dans le premier modèle [Kim14]). 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 du gradient, les gradients de la fonction de coût sont calculés par la règle de dérivation des fonctions composées (*chain rules*). Ainsi, ces gradients sont multipliés successivement. Lorsque ces valeurs sont faibles, le gradient est quasi nul, ainsi les poids ne se mettent plus à jour. À l'inverse, le gradient explose si les gradients sont grands. 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*.

1.3 Récapitulatifs des résultats

TABLE 1 – Résultats sur base de test

Modèles	AUC	Précision	Temps d'entraînement ¹
Bag of words (n=100)	0.74	82%	7min
Embedding (Fast Text) + CNN	0.96	89%	8.2min
Caractères CNN	0.97	92%	32min
Embedding (w2v) + Bi-LSTM + CNN	0.97	92%	12.7min

Les matrices de confusions des modèles Embedding CNN, Caractères CNN et BILSTM sont disponibles en Annexes 23.

1.4 Environnement de travail

Pour rappel, le but du projet est de fournir une analyse sur les méthodes d'interprétabilité pour des modèles utilisés au sein des équipes de la Société Générale Assurances. Il a donc fallu exploiter leur environnement de travail pour faciliter l'utilisation de ce projet au sein de l'entreprise.

C'est pourquoi toute la partie construction, entraînement et interprétabilité a été développé au sein de Kedro, un *framework python* qui facilite et normalise la construction des projets en *data science*. Le principe de Kedro est de pouvoir contenir au sein de *pipeline* les différentes fonctions de transformations des données et d'entraînement des modèles. L'intérêt est de pouvoir reproduire, maintenir et travailler plus facilement tout ce qui a trait aux modèles. De plus, des modules sont disponibles pour enrichir le *package* comme par exemple un outil de visualisation d'un projet Kedro ; qui permet alors de comprendre comment les objets du projet Kedro s'imbriquent entre eux. Ainsi en Annexes 24 est disponible la structure visuelle de ce projet, du chargement des données jusqu'aux phases de test des modèles.

1. Calculé avec le même ordinateur

2 Vers l'interprétabilité

Une fois les différents modèles entraînés, il est possible de les utiliser afin de classifier les avis des utilisateurs d'*Allocine*. Il est possible d'utiliser les différents indicateurs usuels (AUC, Courbe ROC, ...) afin d'obtenir des informations sur la performance des modèles. Cependant, compte tenu de la complexité des modèles mobilisés dans ce projet (ils comportent des millions de paramètres), il est difficile d'obtenir des explications sur les raisons qui ont poussé le modèle à produire une prédiction donnée. Afin d'essayer de comprendre et d'expliquer les prédictions du modèle, des méthodes d'interprétabilité vont être étudiées dans ce chapitre.

2.1 Présentations des méthodologies

2.1.1 Exploiter l'architecture CNN des modèles : GradCAM

2.1.1.1 Class activation map ou CAM

a. Description

Le modèle d'interprétabilité CAM est un modèle introduit par Zhou [Zho+15] en 2015. Il s'agit d'un modèle non-agnostique : contrairement à LIME, il utilise la structure du modèle produisant les prédictions afin de l'expliquer. Ce modèle s'applique uniquement à une classe restreinte des réseaux de neurones à convolution dans un problème de classification. Initialement, ces modèles s'appliquent à la classification d'images. Dans un CNN, les vecteurs obtenus à chaque couche de réseau de neurones à convolution peuvent être considérés comme des caractéristiques ou *feature maps*. Pour une image, ces *feature maps* correspondent à des éléments de l'image (une *feature map* décrira les yeux des personnages d'une photo, une autre le nez etc). Les *feature maps* sont obtenues après chaque couche : en général, les premières couches donnent des *features maps* peu abstraites tandis que les dernières permettent d'avoir des *features maps* abstraites.

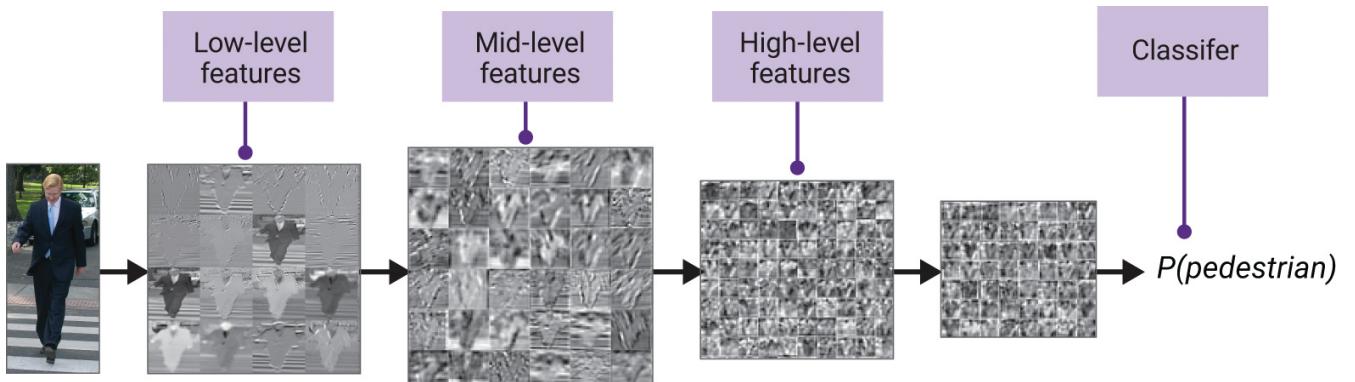


FIGURE 4 – Features maps : issue de <https://www.semiconductorforu.com/artificial-intelligence-impacts-automotive-design/a-cnn-breaks-an-image-into-feature-maps/>

L'idée du modèle CAM est d'essayer d'identifier les éléments de la photographie qui permettent d'expliquer une classe donnée (par exemple chien). Ainsi, certaines *feature maps* permettent de mieux expliquer la probabilité estimée pour une classe : avec l'exemple de la prédiction de la classe "chien", les *feature maps* associées au "museau", "pattes" risque d'avoir une influence plus forte dans la prédiction de cette classe. L'idée de CAM est de fournir une nouvelle *feature map* appelée *Class activation map* qui sera une combinaison linéaire pondérée des *feature maps* d'une couche donnée.

Comme indiqué précédemment, CAM est une méthode qui s'applique uniquement à une famille de CNN. En effet, il faut que le CNN ait l'architecture présentée sur la figure 6 et que la fonction de pooling qui suit la dernière couche convolutionnelle soit une global average pooling (ou GAP). La global average pooling consiste à sommer toutes les composantes d'une *feature map*.

Dans la suite, on suppose qu'on dispose de :

- n *features maps* (f_k) de taille $I \times J$ (I et J sont souvent différents des dimensions initiales de la photographie)
- $F^k = \sum_{i=1}^I \sum_{j=1}^J f_k(i, j)$
- S_c , le résultat avant application de la fonction softmax pour la classe c .

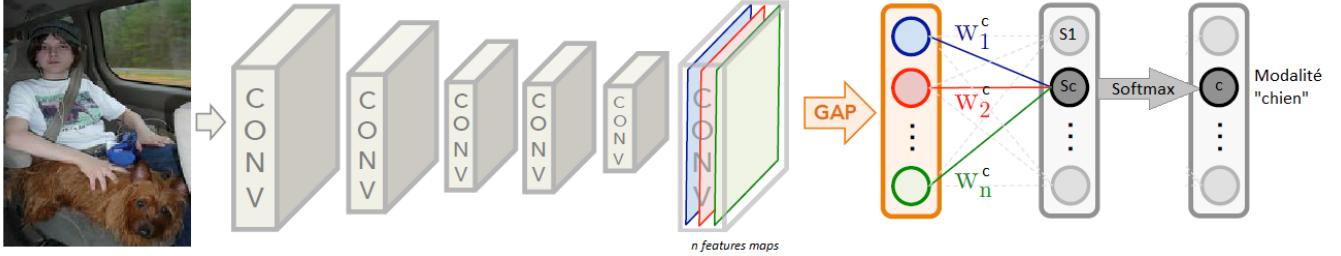


FIGURE 5 – Architecture des modèles acceptés par CAM - Source : [Sel+17]

Il vient que :

$$S_c = \sum_{k=1}^n w_k^c F^k = \sum_{k=1}^n w_k^c \sum_{i=1}^I \sum_{j=1}^J f_k(i, j) = \sum_{i=1}^I \sum_{j=1}^J \underbrace{\sum_{k=1}^n w_k^c f_k(i, j)}_{:=M_c(i, j)} \quad (1)$$

Il s'agit de la classe d'activation pour la classe c au point (x, y)

$$\text{D'où } S_c = \sum_{i=1}^I \sum_{j=1}^J M_c(i, j)$$

M_c peut être appréhendée comme une *feature map* artificielle permettant de quantifier l'intensité de l'activation des *features maps* pour la classe c . Cependant, M_c est une matrice de taille $I \times J$ qui peut être différente de la taille des images initiales. Une opération de suréchantillonage (*upsampling*) est opérée afin de redimensionner M_c à la taille de l'image d'entrée. Les opérations de suréchantillonnages se basent sur des méthodes d'interpolations mais celles-ci ne semblent pas appropriées lorsque les réseaux de neurones à convolution sont utilisés pour la classification de texte.

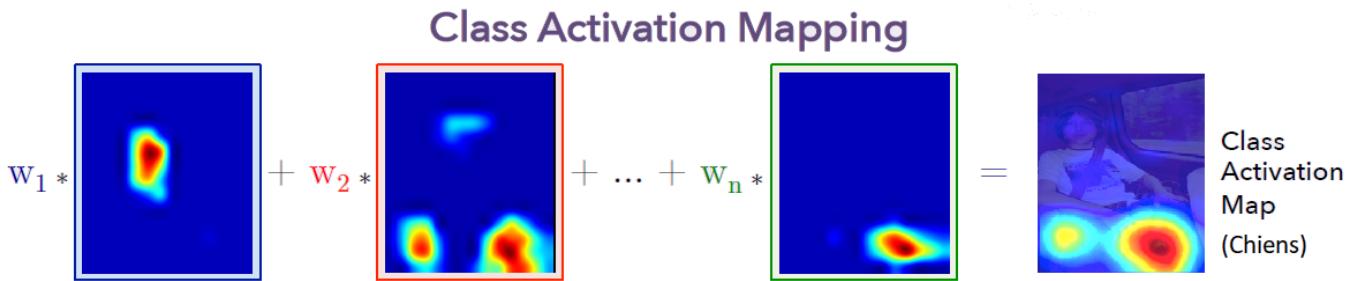


FIGURE 6 – Construction des Class Activation Map - Source : [Zho+15]

b. Inconvénients

Le surmodèle CAM s'applique uniquement à des réseaux de neurones à convolution dont :

- l'avant-dernière couche doit être une couche de Global average pooling précédée par une couche convolutionnelle,
- la dernière couche doit contenir une fonction softmax.

Ces architectures sont très restrictives car elles ne correspondent pas aux réseaux de neurones communément utilisés pour ces tâches. Ainsi Selvaraju proposa en 2017 [Sel+17] une généralisation de cette méthode : Gradient Class Activation Map ou GradCAM.

2.1.1.2 Grad-CAM

Comme vu dans la section précédente, le score pour une classe donnée peut être interprété comme une somme pondérée des *features maps*. Ces poids (correspondant aux poids appris de la dernière couche du réseau) permettent de quantifier l'importance des *features maps* dans la prédiction. Néanmoins, cette relation n'est vraie que si les conditions évoquées à la section précédente sont vraies. Typiquement, cette relation n'est plus vérifiée si la dernière couche de *pooling* est un *1-max pooling* (consiste à prendre le maximum de chaque *features maps*).

GradCam permet de généraliser à une plus large famille de réseaux de neurones convolutionnels l'intuition de CAM. Cet algorithme peut s'appliquer après n'importe quelle couche convolutionnelle (alors que CAM ne s'applique qu'à la dernière couche). Contrairement à l'équation 1, le score n'est plus une fonction linéaire des *features maps* F^k : on supposera, dans la suite, que $S^c = h(F^1, \dots, F^n)$ où h est une application continuement différentiable.

L'idée de Grad-CAM est de créer, comme dans CAM, une *feature map* artificielle sous la forme d'une moyenne

pondérée des features maps sauf que les poids ne seront pas les poids appris à la dernière couche : les poids vont être les dérivées partielles du score pour la classe par rapport aux *features maps*. D'où $M_c^{(g)} := \text{Relu} \left(\sum_{k=1}^n w_k'{}^c F^k \right)$ où $w_k'{}^c = \frac{1}{IJ} \sum_i \sum_j \frac{\partial S^c}{\partial f_k(i,j)}$.

Grad-CAM est une généralisation de CAM dans la mesure où en appliquant la pondération propre à Grad-CAM à un réseau de neurones répondant aux critères de CAM, on retrouve la pondération de CAM.

Comme dans le cas de CAM, une *feature map* artificielle $M_c^{(g)}$ est obtenue mais dont la taille ($I \times J$) n'est pas la même que celle des intrants initiaux : une opération d'*upsampling* est donc nécessaire.

2.1.1.3 Des images aux textes

Les méthodes décrites précédemment sont applicables essentiellement à des réseaux prenant en entrant des images : comment appliquer cette méthode à des modèles de classification textuelle dotés de couche de convolution ?

Le premier problème auquel vont se confronter les modèles qui emploient un *embedding* va être celui du *upsampling* (section 2.1.1.1). En effet, le nombre de mot considéré en entrant est de 67 (après nettoyage des commentaires (section 1.1)). Un commentaire composé de moins de 67 mots se verra donc ajouter autant de vecteur "blanc" (ce mot peut être considéré comme un élément neutre - cette opération se nomme le *padding*) qu'il y a de mots manquants. Au contraire, le commentaire est composé de plus de 67 mots, seuls les 67 premiers seront gardés. Cependant, tronquer les commentaires est problématique pour les longs commentaires car cette opération engendre une grande perte d'information.

Une fois que tous les avis ont la même taille après *padding*, ces derniers peuvent être utilisés dans le réseau de neurones. Cependant, la taille de la *feature map* artificielle obtenue est différente de l'intrant. En effet, l'opération de convolution (passage des filtres) réduit la taille des intrants. Il faut donc effectuer un traitement afin d'obtenir une *feature map* artificielle de la même taille que les intrants. Plusieurs solutions sont possibles : utilisation d'un DeconvNet, utilisation d'autoencoder, méthode de *upsampling*... Ici, la méthode du *upsampling* a été retenue. Cette méthode est la plus basique pour faire correspondre la *feature map* artificielle. Le *upsampling* consiste à interpoler (de manière linéaire par exemple) les composantes d'un vecteur afin d'en ajouter des nouvelles et obtenir un vecteur de taille plus grande. Ces méthodes sont également utilisables sur des matrices comme sur la figure 7.

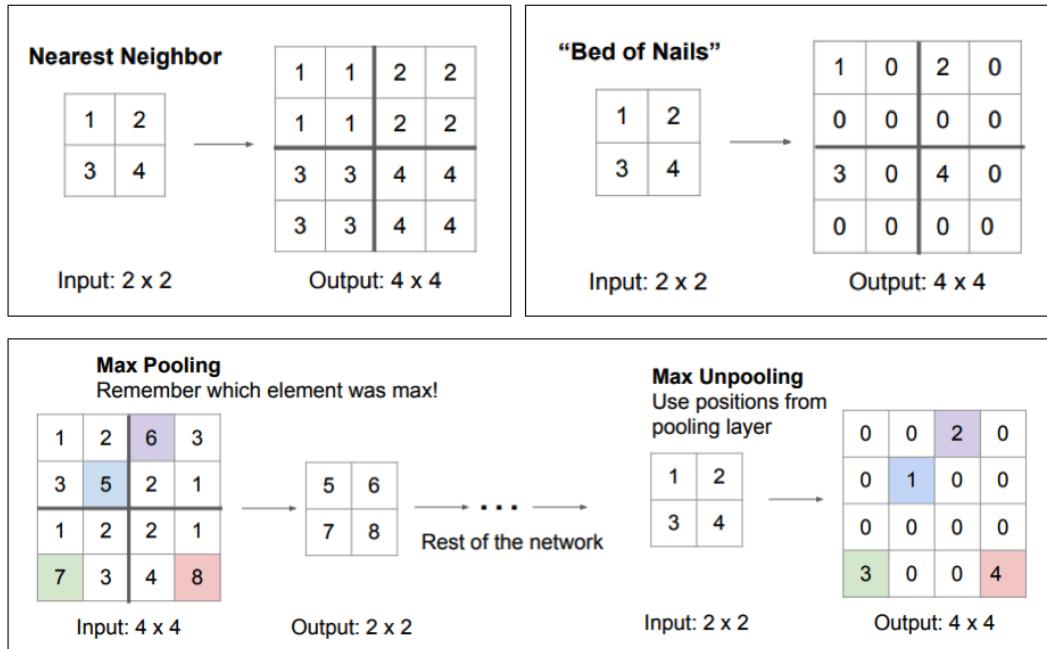


FIGURE 7 – Exemple de méthode d'upsampling - Source : cs231n.stanford.edu

Avec 67 mots uniquement, étirer la *feature map* ne permet pas de la relier avec le commentaire car l'erreur est potentiellement grande.

Sachant que les opérations de convolution et de *pooling* diminue la taille des vecteurs, une solution proposée est de compléter les phrases en intrant du modèle (donc constitué de 67 éléments) par des mots neutres de manière à obtenir après application des convolutions et du *pooling* des vecteurs de taille 67.

Pour le modèle à la maille caractère, la problématique est différente : il n'est pas composé d'une unique couche de convolution. Il faut donc en amont savoir où calculer et récupérer la *feature map*. La couche la plus proche de la sortie contient la plus abstraite des *feature maps* à l'instar des modèles "classiques" de classification d'image (exemple : VGG19 [SZ14]). Le plus intéressant serait de calculer la *feature map* artificielle à partir de cette dernière couche.

Néanmoins, si on utilise la méthode précédente pour étirer de manière factice la *feature map* artificielle (construite à partir de la dernière couche) en augmentant la dimension d'entrée dans le modèle, c'est-à-dire en prenant en compte toutes les étapes de convolution et de *max pooling*, il faudrait passer de 1014 caractères à 9222 (à l'aide du *padding*). Cependant, avec cette dimension, impossible de faire converger le modèle. Il a donc été nécessaire de choisir une couche plus proche de l'entrée du modèle : c'est la première couche qui a été choisie².

Finalement, la *feature map* artificielle générée à partir de la première couche de convolution ne sera pas étirée à l'aide de *padding* mais l'entrant du modèle sera adaptée à la *feature map*, tronquée de quelques derniers caractères.

2.1.1.4 Pseudo-code de GradCAM

Vous pourrez retrouver une version Python utilisant PyTorch dans le répertoire Github du projet³. La solution GradCAM proposée dans ce projet nécessite de modifier la manière de coder le modèle en PyTorch⁴.

Data: un modèle entraîné, un texte, une classe à expliquer

Résultat: la feature map artificielle pour un texte et une classe données

```

initialisation;
prédir la classe du texte avec le modèle entraîné;
enregistrer les gradients par rapport aux feature maps;
enregistrer les feature maps;
for chaque filtre de la couche (si plusieurs tailles il y a) do
    calculer la moyenne des gradients pour chaque feature map;
    scores d'activation ← multiplier chaque feature map avec son gradient moyen associé;
    calculer la moyenne des scores d'activation (entre les features maps);
    feature map artificielle ← scores d'activation moyens normalisés;
end
```

Algorithm 1: GradCAM

2.1.2 Méthodes d'interprétabilité agnostique : LIME et SHAP

2.1.2.1 Méthode agnostique basée sur des surrogates

a. Description

Les *surrogates* sont des modèles d'interprétabilité pouvant être employés sur n'importe quel type de modèles (régression, arbres, réseaux de neurones ...). Ces modèles n'exploitent pas la structure du modèle d'apprentissage mais utilisent directement la prédiction effectuée. Certains modèles peuvent être considérés comme des boîtes noires : énormément de séquences de calculs se suivent empêchant une interprétabilité simple des coefficients estimés. L'idée des *global surrogate* est de fournir un modèle g qui permet de prédire au mieux les prédictions du modèle initial \hat{y} (et non les variables à expliquer y) à l'aide d'un nombre réduit de variables interprétables. Si on considère un modèle $\hat{f}: \mathcal{X} \rightarrow \mathcal{L}$ l'idée est de fournir un modèle $g: \mathcal{X}' \rightarrow \mathcal{L}$ $x \mapsto y$, $x' \mapsto \hat{y} = \hat{f}(x)$, \mathcal{X}' correspond à un ensemble de caractéristiques dites "interprétables".

2. Toutes les autres couches n'ont pas permis de faire converger le modèle

3. https://github.com/ENSAE-CKW/nlp_understanding/blob/master/src/deep_nlp/grad_cam/model/gradcam_base_model.py

4. En Annexe [1] et [2] est disponible un exemple de la solution avant/après GradCAM pour le modèle BiLSTM CNN

Exemple de *global surrogate* sur un classifieur de sentiments

L'exemple pris ici est un modèle de classification de sentiments^a basé sur le jeu d'observations (x_i, y_i) , où x_i est un vecteur de taille s et $y_i \in \{1, \dots, M\}$.

Un réseau de neurones convolutionnel f est entraîné afin d'obtenir une prédiction. Les prédictions par ce réseau sont notées $\hat{y}_i = f(x_i)$. Bien que les performances des CNN pour la classification de sentiments soient remarquables [Kim14], ces modèles sont peu interprétables. Il est possible d'utiliser un *global surrogate* où les variables interprétables seraient des indicatrices de la présence de certains mots ou groupes de mots à l'aide d'un modèle facilitant l'interprétation (ces variables permettant d'expliquer sont choisies *a priori* et peuvent être différentes des intrants du réseau de neurones).

Le réseau de neurones va prédire la classe "Positif" pour la phrase "Je suis très content de ce film, il s'agit d'un chef d'oeuvre". Le *global surrogate* peut être ici un modèle logistique classique où les variables explicatives seront des indicatrices de mots-clefs ("bien" "content" "nul" "intéressant" ...) et la variable à expliquer sera la prédiction (et non le *label*) du sentiment de la phrase réalisée par le réseau de neurones.

a. Le but est de déterminer si un texte est plutôt positif ou négatif.

b. Avantages et inconvénients

Les *global surrogate* sont une famille de modèles d'interprétabilité très intuitifs : à partir des prédictions d'un modèle difficilement interprétable, un modèle plus simple est entraîné en utilisant des variables plus explicites. Ces méthodes sont agnostiques : elles s'adaptent à n'importe quel type de modèles. Cette propriété est très appréciable car il est possible de modifier le modèle initial sans avoir à modifier le modèle d'interprétabilité.

Néanmoins, ces modèles nécessitent donc d'apprendre un deuxième modèle ajoutant potentiellement des sources d'erreurs supplémentaires. Si l'explication n'est pas satisfaisante, il est difficile de savoir si la source de ce problème provient du modèle initial ou du sur-modèle. De plus, ils proposent des interprétations globales : il est possible que localement, l'interprétation fournie soit très mauvaise. Pour répondre à cette problématique, des méthodes de *surrogate* locaux ont été développées.

2.1.2.2 Local Interpretable Model-agnostic Explanations (LIME)

a. Description

Comme dit précédemment, les *global surrogate* permettent une interprétabilité globale : il se peut que les explications fournies pour certaines prédictions ne soient pas pertinentes. Pour remédier à ce problème, Ribeiro propose une approche locale nommée LIME (pour *Local Interpretable Model-agnostic Explanations*) [RSG16]. L'idée de LIME est d'apprendre pour chaque prédiction un *surrogate* au voisinage de l'exemple considéré. Ce *surrogate* devra répondre aux mêmes contraintes que dans le cas global : utiliser des variables explicites et être facilement interprétable. Contrairement aux *global surrogates*, LIME apprendra un modèle d'interprétabilité en utilisant un jeu de données artificiel : il s'agit de réplications bruitées de l'exemple considérée (par exemple, pour une variable binaire, la valeur pourra être inversée avec une certaine probabilité).

Concrètement, pour un couple d'observations (x, y) , LIME réalise les étapes suivantes :

1. Choisir une observation x et calculer les variables interprétables associées x' .
2. Créer le jeu de données des variables interprétables bruitées \mathcal{Z} de x .
3. Calculer les prédictions $\hat{y}_{\mathcal{H}}$ par la modèle "boîte noire" pour chaque observation du jeu de données bruitées⁵.
4. Pondérer le jeu de données par la similitude entre observation perturbée z et observation initiale x' . Cette similitude sera quantifiée à l'aide d'un noyau.
5. Entrainer à l'aide du jeu de données \mathcal{Z} , un modèle interprétable prédisant $\hat{y}_{\mathcal{H}}$ à l'aide de \mathcal{Z}

5. On suppose donc qu'il est possible de passer des variables initiales x aux variables interprétables x' et vice versa.

Exemple d'utilisation de LIME sur un classifieur d'images

L'exemple pris ici est un modèle de classification d'images avec M classes contenant "Guitare électrique", "guitare acoustique" et "Labrador" basé sur le jeu d'observations (x_i, y_i) où x_i est une image (décrise par un tenseur de taille $H \times L \times 3$ et $y_i \in \{1, \dots, M\}$). L'algorithme fournit des probabilités d'appartenance pour chacune des M classes.

Appliquons l'algorithme LIME à l'image (a) de 8 :

1. Il faut commencer par définir des variables interprétables. En effet, la photo est décrise par un tenseur de taille $H \times L \times 3$ donc par $3HL$ valeurs comprises entre 0 et 1. Cette description ne permet pas d'interpréter simplement les sorties (il est difficile de se représenter une de ses $3HL$ valeurs). Des variables interprétables ici peuvent être des superpixels : il s'agit d'agrégations de pixels ayant des similitudes. Les superpixels permettent de découper l'image en P morceaux. L'intensité de ces superpixels sera une agrégation des intensités des pixels le constituant (par exemple, la moyenne). Sur l'image (d), on identifie deux superpixels.
2. Il faut maintenant constituer un jeu de données bruité à partir de la version *superpixel* de l'image. Pour cela, on peut par exemple "éteindre" certains superpixels tirés de manière aléatoire^a en réduisant leur intensité à 0.
3. Pour chaque nouvelle image, le réseau de neurones fournit la probabilité d'appartenir à une classe donnée.
4. Puis un modèle plus interprétable (comme un modèle de régression LASSO) est appris pour fournir les prédictions obtenues à l'étape précédente à partir des nouvelles images.
5. Les coefficients de ce modèle plus interprétable permettent d'identifier les superpixels expliquant au mieux la prédiction du réseau de neurones.

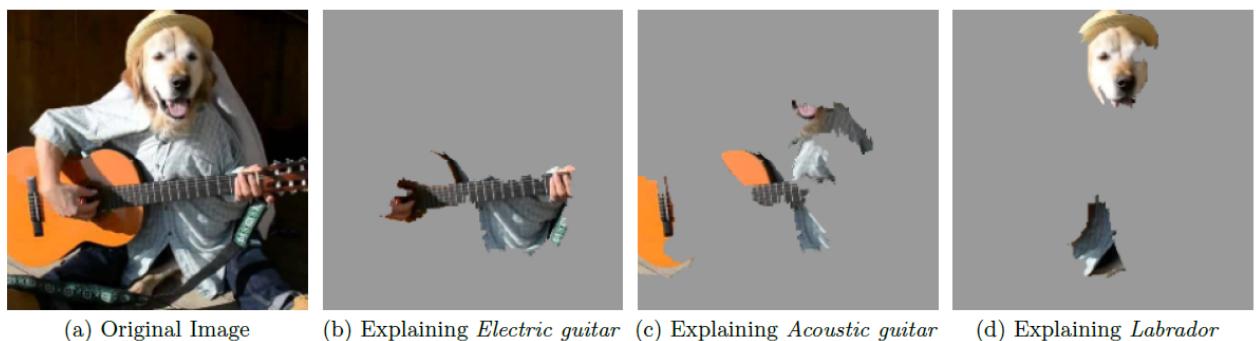


FIGURE 8 – Illustration de LIME pour la classification d'image. Source : Rebeiro (2016)

a. En tirant de manière indépendante pour chaque superpixel des variables de Bernouilli

b. Application à des données textes

Dans le cadre de ce projet, l'algorithme LIME a été appliqué à des problèmes de classifications où les variables explicatives étaient des phrases. Les variables interprétables sont des indicatrices de la présence de certains mots ou couples de mots dans la phrase. Afin d'obtenir le jeu bruité au voisinage de la phrase considérée, il suffit de retirer aléatoirement certains mots⁶.

Le modèle entraîné (cf partie 1.2) est ensuite utilisé pour prédire la probabilité d'appartenance aux catégories. Puis, pour une catégorie donnée, un modèle plus interprétable est entraîné afin d'obtenir les estimations du modèle à partir des indicatrices de certains mots dit interprétables. Les coefficients donnent ainsi l'importance de chaque mot dans la prédiction d'une catégorie donnée.

Il est à noter que le *surrogate* est appris en utilisant des poids : plus une observation est *perturbée*, plus son poids est faible. Formellement ce poids est donnée par la formule suivante : $K_\gamma(D(x, x'))$ où D est une distance sur l'espace des variables interprétables et K un noyau. Dans l'implémentation Python de la bibliothèque `lime`, le noyau⁷ utilisé par défaut est le noyau gaussien ($K_\gamma(x) = \exp(-\gamma x^2)$), la mesure de similitude est la similitude cosinus et le sur-modèle est une régression ridge. Cependant, nous avons décidé d'utiliser une régression linéaire multiple à la place car la régression ridge n'apporte pas de *sparsité* (donc ne facilite pas forcément l'interprétation) et apporte un hyperparamètre supplémentaire (intensité de la pénalisation L^2).

6. Comme précédemment, on peut utiliser des tirages indépendants de variable de Bernouilli pour chaque mot certains mots de la phrase

7. Un noyau K est une application de \mathbb{R} dans \mathbb{R}^+ tel que $\int_{\mathbb{R}} K(x)\lambda(dx)$

2.1.2.3 Méthode SHAP

a. La valeur de Shapley

La valeur de Shapley est un concept issu de la théorie des jeux. Elle permet de calculer la contribution du i-ème joueur (de la i-ème variable dans notre cas). Posons :

- F l'ensemble des variables
- $\mathbb{E}[f(X) | X_S = x_S]$ l'espérance conditionnelle de la fonction de prédiction du modèle quand un sous-ensemble S de variables est fixé au point x

On a alors la valeur de Shapley associé à une observation x et un modèle :

$$\phi_i(f, x) = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} (\mathbb{E}[f(X) | X_{S \cup \{i\}} = x_{S \cup \{i\}}] - \mathbb{E}[f(X) | X_S = x_S])$$

Ainsi, l'idée est de sommer, pour chaque sous-ensemble de l'ensemble des variables, l'écart de prédiction avec la variable i et sans elle. Il est à noter que la valeur de Shapley est additive, c'est-à-dire qu'on a pour une observation x :

$$f(x) = E(f(X)) + \sum_i \phi_i(f, x)$$

Ainsi cette valeur explique comment chaque variable contribue à écarter la prédiction du modèle de la moyenne des prédictions. Ceci est un avantage de l'utilisation de cette valeur de Shapley. Cependant, le calcul de cette valeur est très coûteux à cause notamment du nombre exponentiel de sous-ensemble de variables $2^{|F|}$. De plus, il est difficile d'estimer les espérances conditionnelles.

Pour remédier à ces difficultés, il existe plusieurs algorithmes. Nous allons ici nous intéresser à l'un d'entre eux : Kernel SHAP.

b. L'algorithme Kernel SHAP

Cet algorithme est agnostique au modèle. Il est basé sur une extension de LIME. En fait, en choisissant correctement les paramètres de LIME (fonction de coût, noyau et complexité), les coefficients du modèle interprétable g sont une très bonne approximation des valeurs de Shapley. Avec cet algorithme, l'absence d'une variable est simulée par le remplacement de sa valeur par la valeur prédominante qu'elle prend dans un « background dataset » (demandé en entrée de l'algorithme).

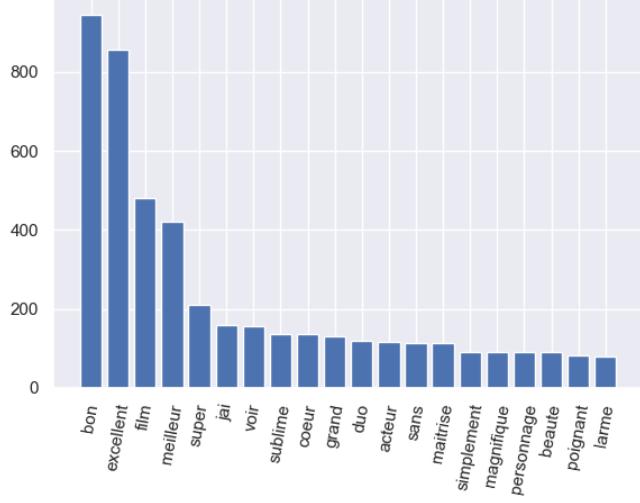
2.2 Comparaison des résultats

2.2.1 Résultats du GradCAM au global

Dans un premier temps, ces méthodes d'interprétabilité vont être testé de manière global : elles seront donc appliquées à l'ensemble des données de test puis les résultats seront agrégés.

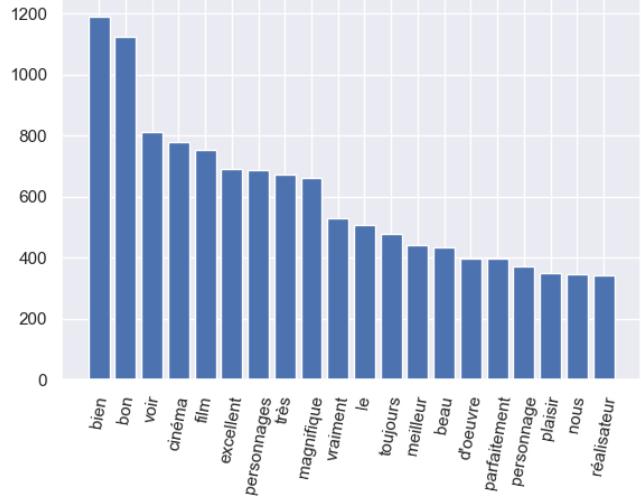
Comme expliqué précédemment, la méthode GradCAM permet de créer une *feature map* artificielle, qui met en avant pour une classe donnée, l'importance d'un entrant (ici, l'importance d'un mot). Arbitrairement, nous avons décidé que, parmi tous les inputs, les plus importants (pour une classe donnée) seraient ceux tels que la valeur au sein de la *feature map* artificielle soit supérieure à seuil. On peut à présent compter les mots qui ont une valeur supérieure à ce seuil pour pouvoir vérifier si par exemple, pour la classe 1 (commentaire positif), les mots les plus importants nous semblent positifs. C'est ce que nous avons représenté pour chaque modèle (Figure 9) pour les commentaires classifiés positifs (probabilité supérieure à 0.75). En Annexe 25, nous les avons représenté pour la classe 0 pour les commentaires prédits négatifs

Explication globale pour la classe 1 pour les probabilités les plus élevées (≥ 0.75)



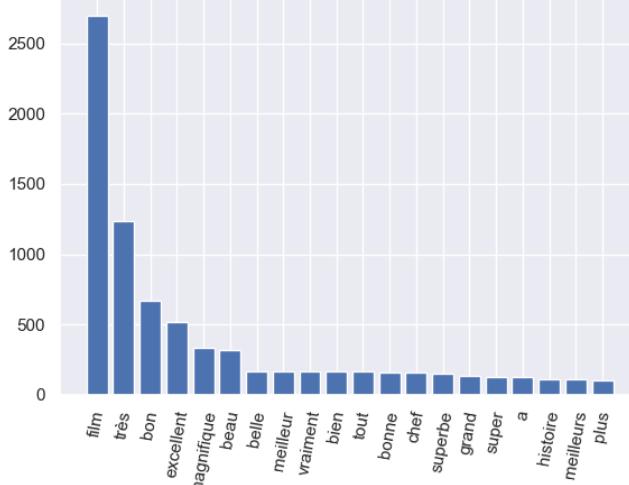
(a) Vanilla CNN

Explication globale pour la classe 1 pour les probabilités les plus élevées (≥ 0.75)



(b) Caractères CNN

Explication globale pour la classe 1 pour les probabilités les plus élevées (≥ 0.75)



(c) Bi-LSTM CNN

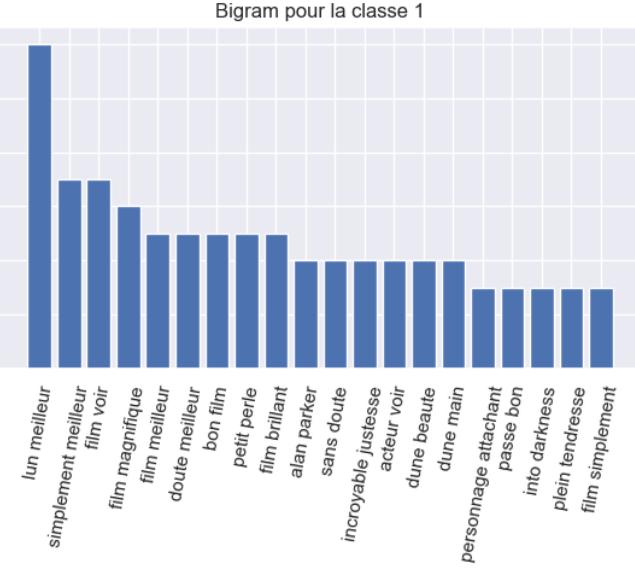
FIGURE 9 – Fréquence des mots les plus importants pour la classe 1, pour les commentaires classifiés positifs

Pour chacun des trois modèles, des mots comme "excellent", "bon", "magnifique" apparaissent souvent comme important pour prédire un commentaire comme étant positif. Néanmoins, on retrouve aussi des mots génériques comme "film", "jai"/"je", "a" ou encore "très". Ils sont génériques car ils apparaissent aussi pour la classe 0 en Annexe 25. Cependant, ce n'est pas anormal, car par exemple un film peut-être "vraiment très nul" ou "très impressionnant".

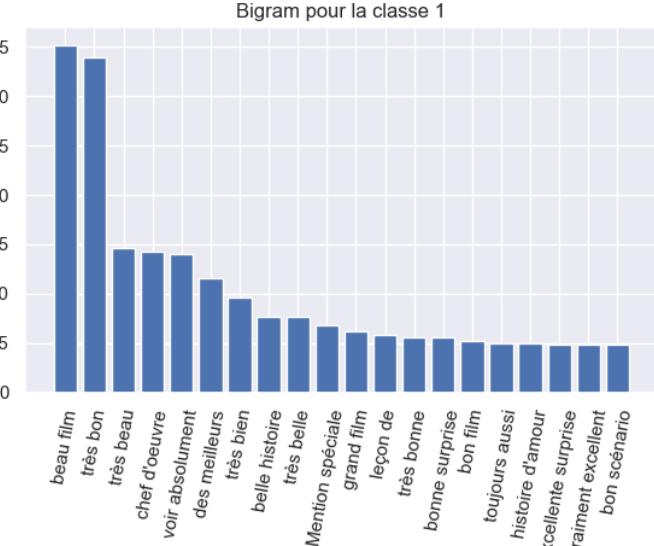
Ces mots, qu'on retrouve à la fois dans l'importance de la classe 0 et 1 pour tous les modèles, sont des mots qui peuvent être suivis ou précédés d'autres mots qui, eux, peuvent signaler le caractère positif ou négatif d'un commentaire. Ainsi, ils sont importants car les filtres des couches de convolution (selon leurs tailles) vont conjuguer le sens des mots porteurs du sentiment d'un commentaire sur ces mots génériques. Une partie du signal est donc distribuée à ces mots.

Pour mettre en exergue ce phénomène sont considérés des *bi-gram* d'importance. Nous avons défini un *bi-gram* d'importance comme une suite de deux mots considérés comme important (i.e. deux mots consécutifs qui ont une valeur au sein de la *feature map* artificielle supérieure à un seuil). Puis, nous avons compté ces *bi-gram* d'importance de la même manière que dans la Figure 9.

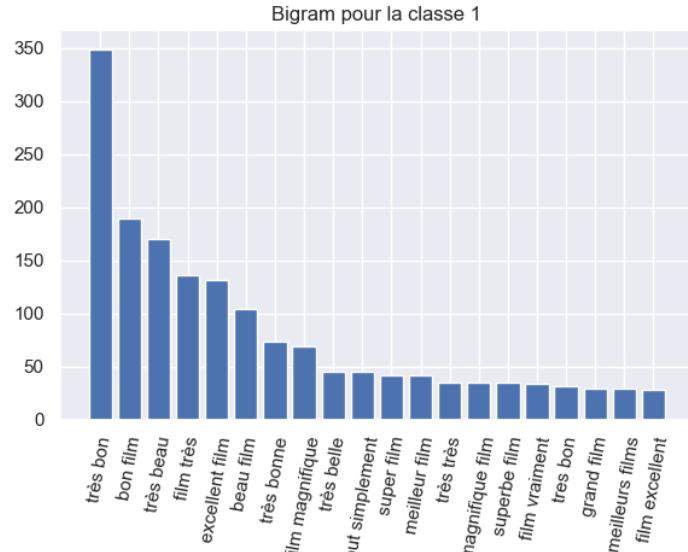
C'est ce que nous avons représenté Figure 10 pour la classe 1 pour les commentaires prédis positifs. De façon analogue en Annexe 26 vous pourrez retrouver les *bi-gram* pour la classe 0.



(a) Vanilla CNN



(b) Caractères CNN



(c) Bi-LSTM CNN

FIGURE 10 – Fréquence des *bi-gram* les plus importants pour la classe 1, pour les commentaires classifiés positifs

Remarques :

- Pour un même seuil, les *bi-gram* ne sont pas aussi fréquents selon les modèles. En effet, pour les vingt *bi-gram* les plus courants, celui qui apparaît le plus souvent pour le modèle Vanilla CNN est "jai adorer" (environ trente-cinq fois) contre près de quatre-cents pour le *bi-gram* le plus fréquent pour le modèle Bi-LSTM CNN "bon film".
- Deux modèles, Caractères CNN et Bi-LSTM semblent mettre en avant des *bi-gram* similaires.

Ainsi, le mot d'importance "très" qui était observé à la fois dans la Figure 9 et dans l'Annexe 25 se retrouve dans les *bi-gram* d'importance sous la forme "très bon"/"très beau"/"très bien" dans la Figure 10 et sous la forme "très mauvais" dans l'Annexe 26. De plus, le mot "film" est associé au mot "excellent" dans la Figure 10, et "ennuyeux" dans la Figure 26. Par conséquent, l'hypothèse initiale qui sous-tendait l'idée que certains mots jugés importants se retrouvaient à la fois dans l'explication de la classe 0 et 1 parce qu'ils étaient entourés d'un mot porteur du sens, semble se vérifier.

A priori, la méthode GradCAM semble mettre en avant pour une classe donnée des mots qui semblent cohérents avec la classe prédite par les modèles lorsqu'elle est testée sur l'ensemble des données de test. Donc au global, les résultats sont satisfaisants. Par la suite, vont être traitées les questions d'interprétabilité locale, c'est à dire au niveau du commentaire.

2.2.2 Exemples atypiques

Contrairement à un modèle d'apprentissage, il est très délicat d'estimer la performance d'un modèle d'interprétabilité. En effet, quel indicateur ou quel algorithme peut être mis en oeuvre afin de quantifier le pouvoir d'interprétabilité ? Pour tenter de l'évaluer pour les différentes méthodes, celles-ci vont être appliquées à certains exemples. Ces exemples vont être groupés en quatre catégories :

- des exemples classés positivement par le modèle (probabilité d'appartenance à la classe "Positif" proche de 1) et dont l'étiquette est "Négatif".
- des exemples classés négativement par le modèle (probabilité d'appartenance à la classe "Positif" proche de 0) et dont l'étiquette est "Positif".
- des exemples où un modèle prédit très mal alors que les deux autres modèles prédisent correctement
- des exemples pour lesquels le modèle a eu des difficultés à classer (probabilité d'appartenance à la classe "Positif" environ de 0.5)

Dans les paragraphes suivants, les résultats des prédictions pour un exemple donné seront décrits à l'aide d'un tableau de la forme :

Phrase	CNN Char	CNN Embed	BiLSTM Embed	Label
"Vive Gradcam, c'est vraiment le feu comme méthode"	0.95	0.9	0.1	1

TABLE 2 – Phrase : Phrase dont les prédictions sont effectuées

CNN Char : Prédiction de la probabilité d'appartenance à la classe "Positif" par le modèle à la maille caractère utilisant un CNN

CNN Embed : Prédiction de la probabilité d'appartenance à la classe "Positif" par le modèle à la maille mot utilisant un CNN

BiLSTM Embed : Prédiction de la probabilité d'appartenance à la classe "Positif" par le modèle à la maille mot utilisant un réseau BiLSTM suivi d'un CNN

Label : Étiquette de l'observation

Enfin voici comment interpréter les graphiques qui vont suivre :

- **GradCAM** : plus la couleur d'un mot est verte foncée, plus ce mot est important pour le modèle pour une classe donnée.
- **LIME** : plus la couleur d'un mot est orange foncée (respectivement bleue foncée), plus ce mot est important pour la classe 1 (respectivement la classe 0).
- **SHAP** : plus la couleur d'un mot est verte foncée (respectivement violette foncée), plus ce mot est important pour la classe 1 (respectivement la classe 0).

A noter que les méthodes SHAP et LIME ne sont pas appliquées au modèle CNN Char car les variables d'entrée de ce modèle sont des caractères. Ainsi, substituer des caractères pour faire varier la sortie du modèle n'a pas vraiment de sens.

2.2.2.1 Exemples bien classés par un modèle mais pas par les autres

L'exemple suivant a été classé positivement par le modèle CNN mais négativement par le modèle Bi-LSTM :

Phrase	CNN Char	CNN Embed	BiLSTM Embed	Label
un film positif et des enfants attachants, enfin une goutte d'espoir menée par un Kad Merad fascinant utilisé à contre-emploi	0.49	0.99	0.00	1

TABLE 3

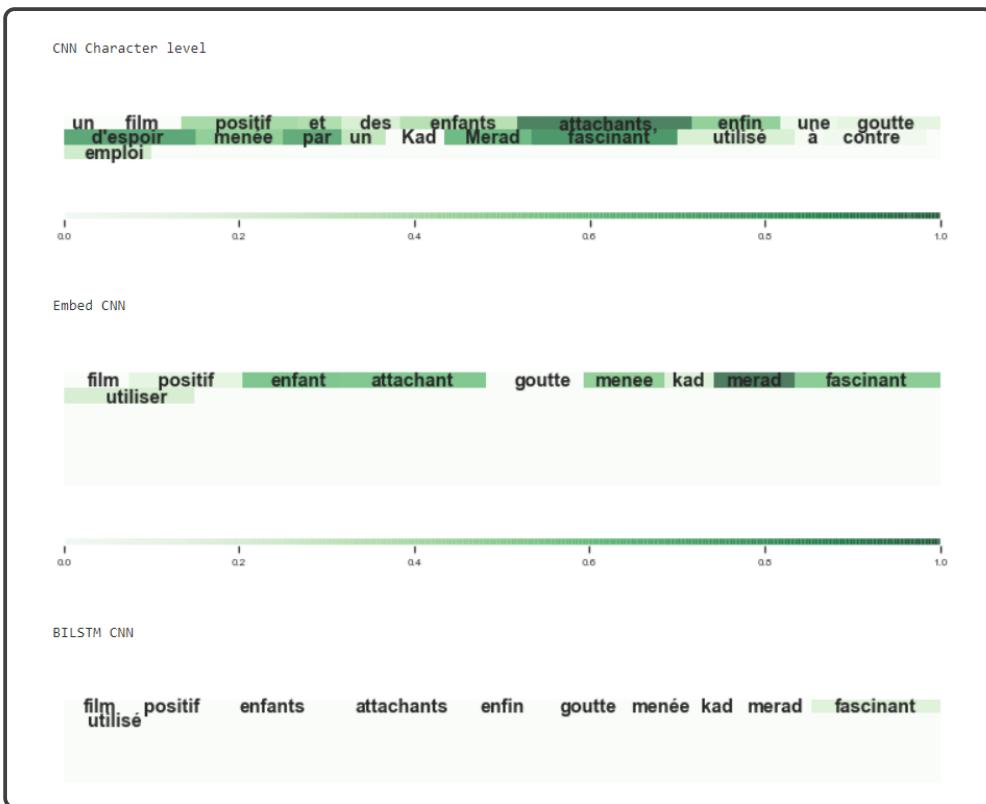


FIGURE 11 – Application de GradCAM pour la classe 1 sur l'exemple 516

GradCAM Dans cet exemple, le GradCAM pour la classe 1 ne met rien en avant pour le BiLSTM (quant à la classe 0 (Annexe 27) les résultats sont très étonnantes). Pour les modèles Embedding CNN et à la maille caractère, les résultats sont intéressants, mais le mot "merad" qui ressort est intrigant.

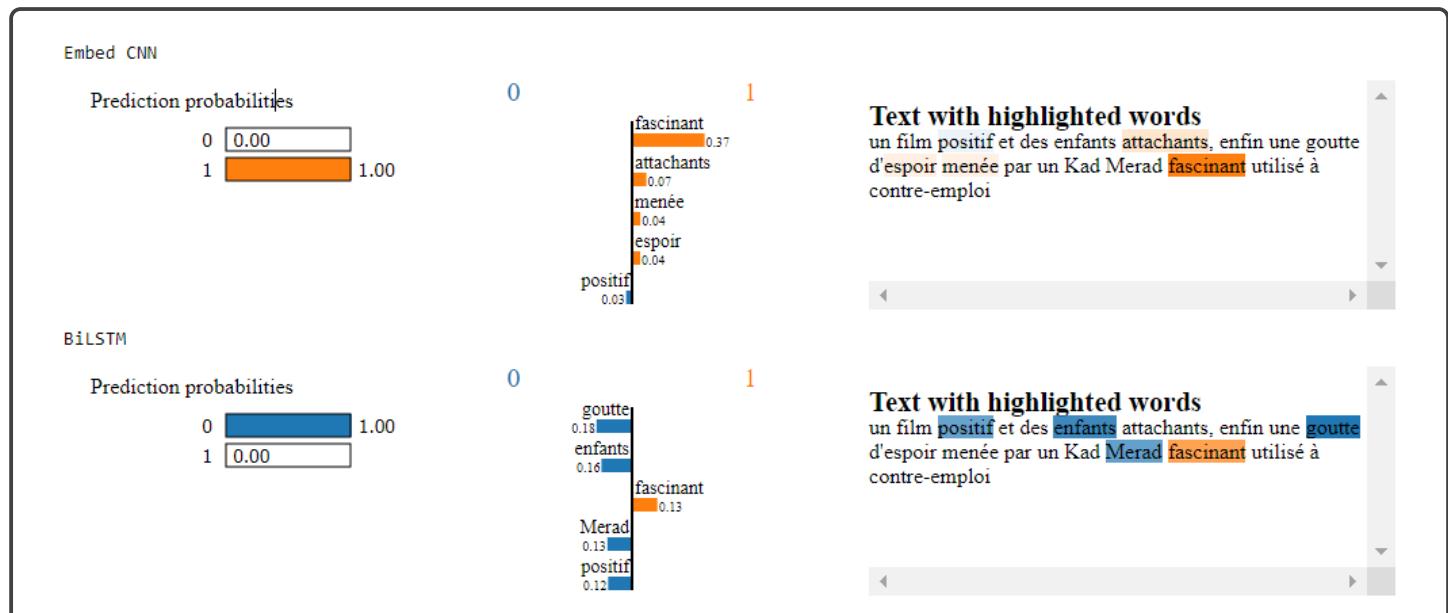


FIGURE 12 – Application de LIME sur l'exemple 516

LIME Dans les deux cas, le modèle LIME ne permet pas d'expliquer correctement pourquoi ces prédictions ont été réalisées. LIME semble moins bien fonctionner sur le modèle BiLSTM : cela s'entend dans la mesure où la prédiction est très mauvaise.

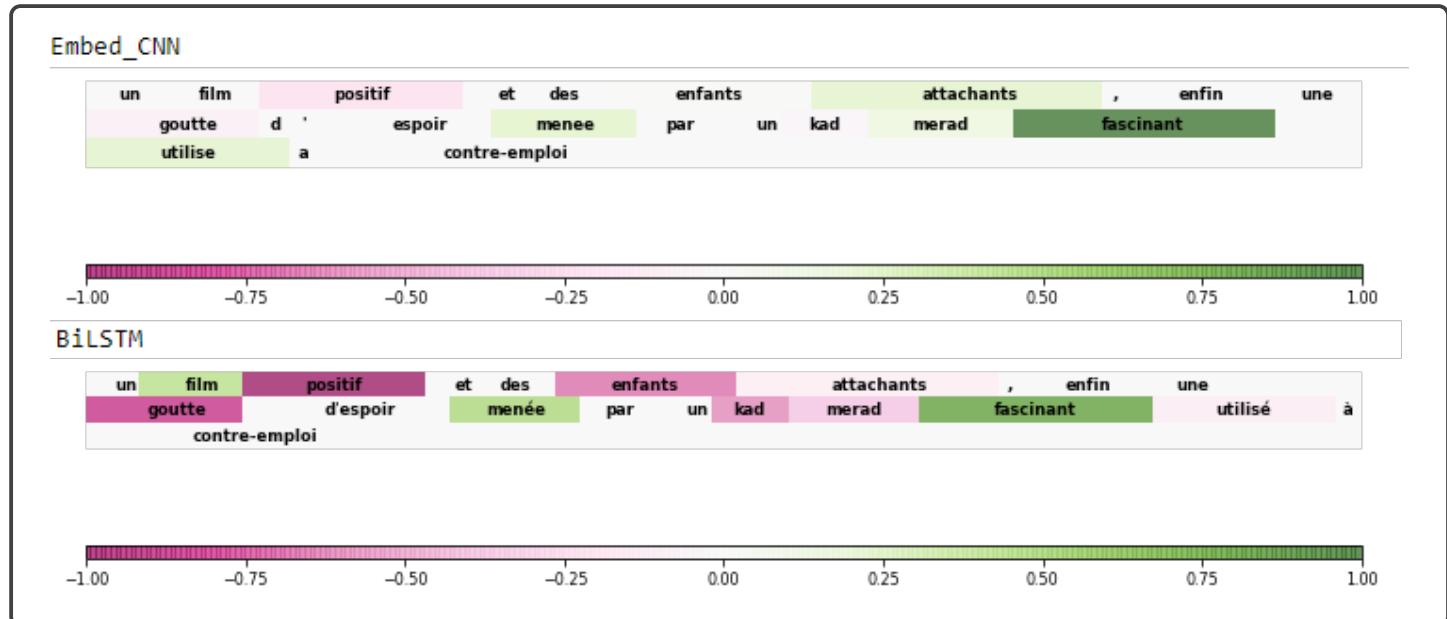


FIGURE 13 – Application de SHAP sur l'exemple 516

SHAP Pour le modèle embedding CNN, SHAP met en avant la forte contribution du mot "fascinant" qui expliquerait presque à lui seul la prediction du modèle. Pour le modèle BiLSTM, les résultats semblent mauvais, une nouvelle fois probablement car la prédiction de ce modèle n'est pas bonne.

L'exemple suivant a été bien classé par le modèle BiLSTM mais mal par le modèle CNN.

Phrase	CNN Char	CNN Embed	BiLSTM Embed	Label
Un très beau film de Gérard Jugnot.				
Un très bon drame sur l'occupation allemande en France.	0.99	0.04	0.99	
Un sujet dur bien traité.				1

TABLE 4

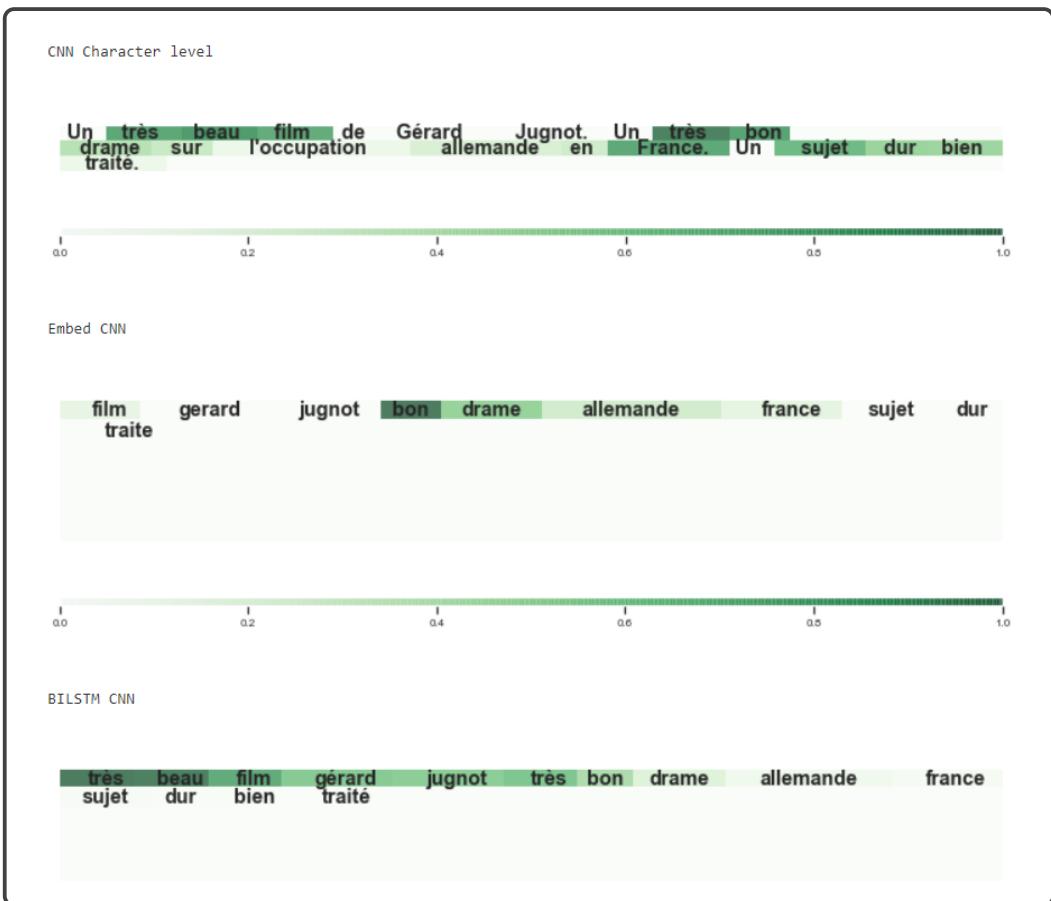


FIGURE 14 – Application de GradCAM pour la classe 1 sur l'exemple 8619

GradCAM Pour le modèle Embedding CNN, le mot "bon" est logiquement mis en avant pour la classe 1, mais il ne semble pas avoir joué un grand rôle dans la prédiction de ce commentaire. Au contraire pour la classe 0 (en Annexe 28) c'est "gérard" et "jugnot" qui sont mis en avant. Pour les deux autres modèles, la méthode semble fonctionner.

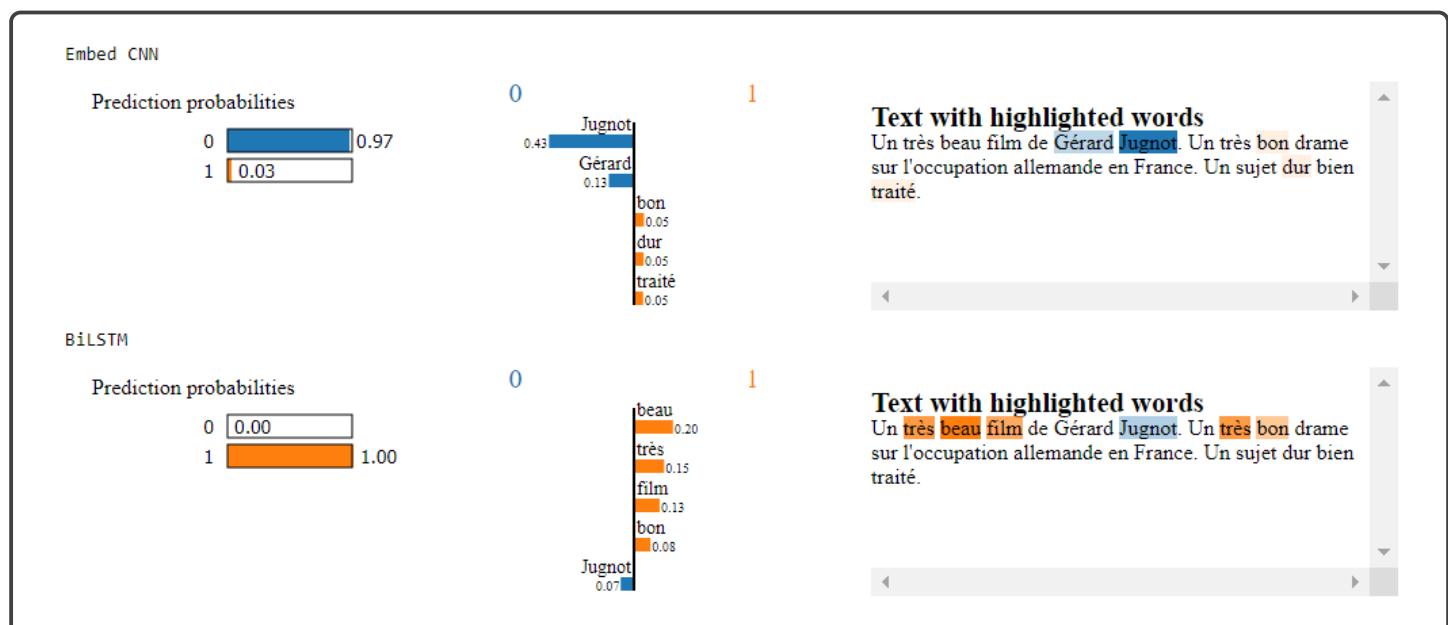


FIGURE 15 – Application de LIME sur l'exemple 8619

LIME Cette fois, LIME fonctionne bien dans le cas où la prédiction est correcte (BiLSTM) mais ne fonctionne pas du tout dans le cas CNN.

Premier conclusion (réalisée sur plus d'exemples) : LIME fonctionne lorsque la prédiction est correcte. LIME ne semble pas être mobilisable pour essayer d'expliquer une mauvaise prédiction

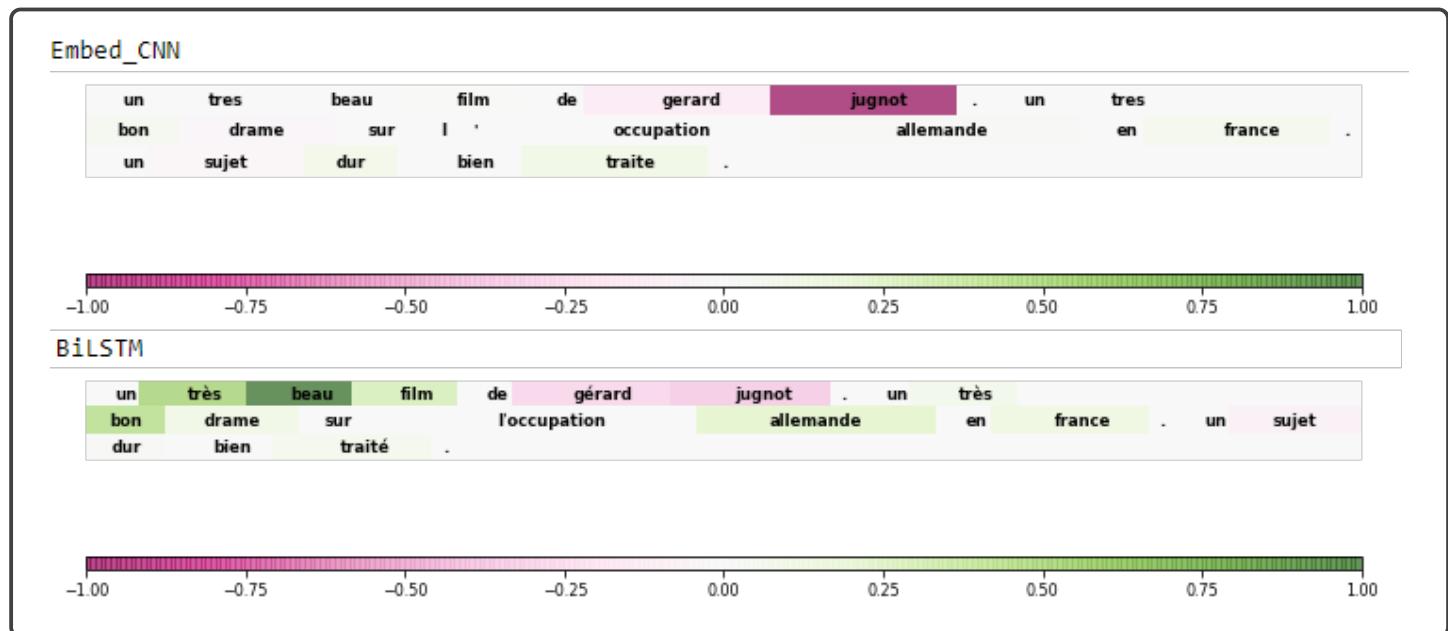


FIGURE 16 – Application de SHAP sur l'exemple 8619

SHAP SHAP explique la très mauvaise prédiction du modèle embedding CNN par la présence de "Jugnot". Le résultat est bon ici pour le modèle BiLSTM.

2.2.2.2 Exemples mal classés par tous les modèles

L'exemple suivant a été mal classé par l'ensemble des modèles :

Phrase	CNN Char	CNN Embed	BiLSTM Embed	Label
Domage que les derniers 3/4 heure du film de soit pas aussi bon que le debut et milieux du film.				
Quelques carence égale scénaristique avec des enchainement un peu facile, téléphoné et pas crédible. Scène au honduras avec les indient et les bonne soeur totalement inutile...	0.011	0.0016	0.056	1

TABLE 5

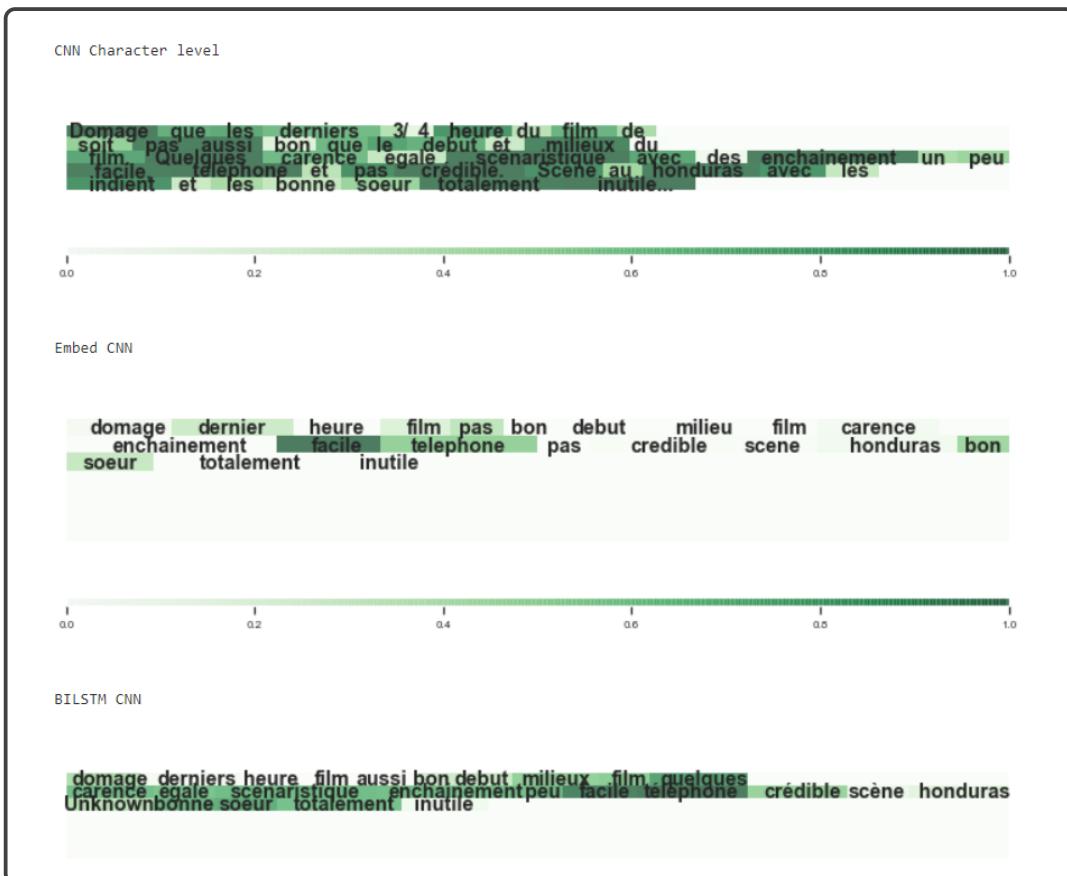


FIGURE 17 – Application de GradCAM pour la classe 0 sur l'exemple 1287

GradCAM Pour le modèle à la maille caractère, en raison de la méthode d'agrégation⁸, le résultat est confus. Pour le modèle Embedding CNN, les résultats sont intéressants, avec la négation mise en avant. Enfin le BiLSTM, les explications amenées ne sont pas probants.

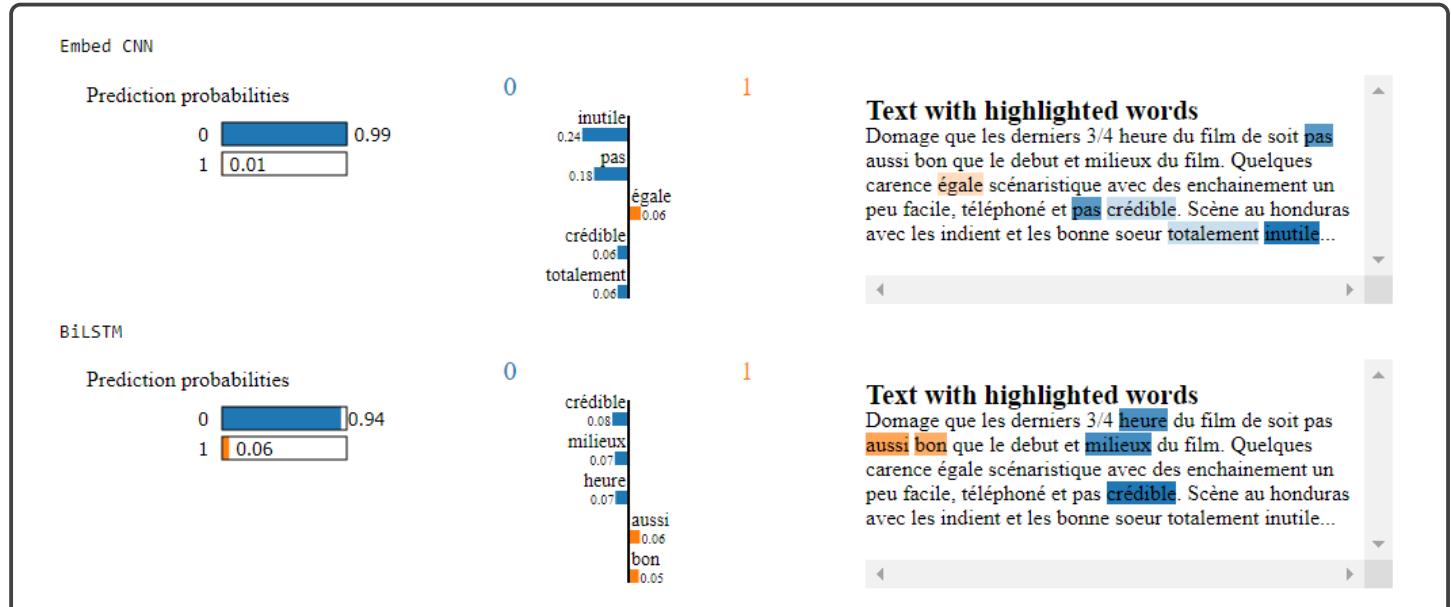


FIGURE 18 – Application de LIME sur l'exemple 1287

LIME Ici, l'exemple est mal libellé (il s'agit d'une critique négative du film étiquetée comme positive). LIME permet de justifier la prédiction par le modèle CNN en négatif (par exemple avec les mots "pas", "crédible" et

8. Pour le modèle à la maille caractère, la méthode GradCAM fournit une *feature map* artificielle associée aux caractères et non aux mots. Il est donc nécessaire d'aggrégier l'information au niveau des mots.

"inutile"). Pour autant, les explications apportées pour le modèle BiLSTM ne semble pas concluante.

Encore une fois, LIME semble moins bien fonctionner pour le modèle BiLSTM

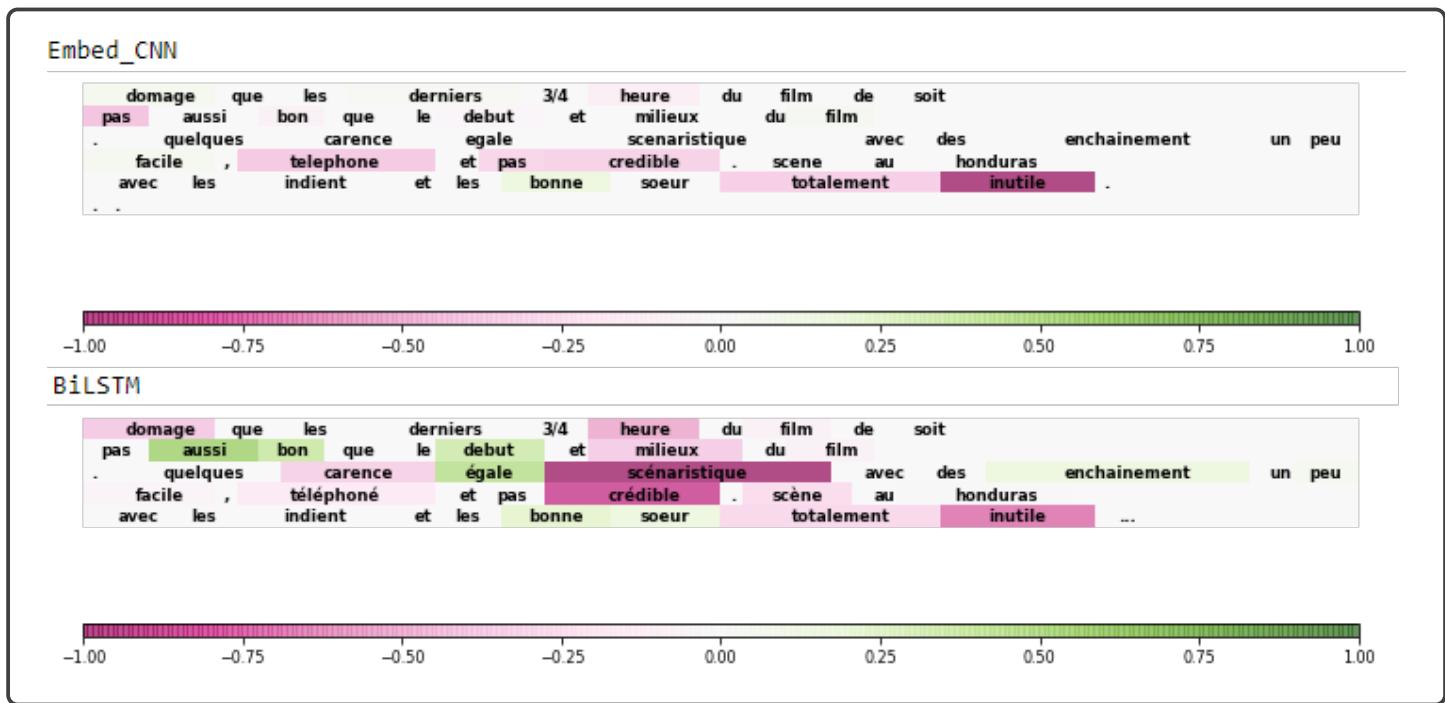


FIGURE 19 – Application de SHAP sur l'exemple 1287

SHAP SHAP fonctionne bien ici pour le modèle Embedding CNN. il arrive aussi à expliquer la prédiction de BiLSTM (avec notamment "domage", "inutile" ou "crédible" qui est précédé de "pas"), mais il donne des résultats étonnantes (notamment "scénaristique" est le mot plus influant).

2.2.2.3 Exemples où le modèle hésite (prédiction proche de 0.5)

Phrase	CNN Char	CNN Embed	BiLSTM Embed	Label
Je pense qu'il faut vraiment être amoureux de cette ville pour aimer ce film. Pour ma part je n'ai pas accroché mais je reconnaît que ce film possède certaines qualités, notamment sa musique qui est souvent intéressante. Ce film est essentiellement fait d'images d'archives.				
Pour ma part je n'ai pas accroché mais je reconnaît que ce film possède certaines qualités, notamment sa musique qui est souvent intéressante. Ce film est essentiellement fait d'images d'archives.	0.68	0.5	0.07	0
Il faut aimer car je n'ai pas trouvé de vrai fil conducteur... mais les goûts et les couleurs!				

TABLE 6

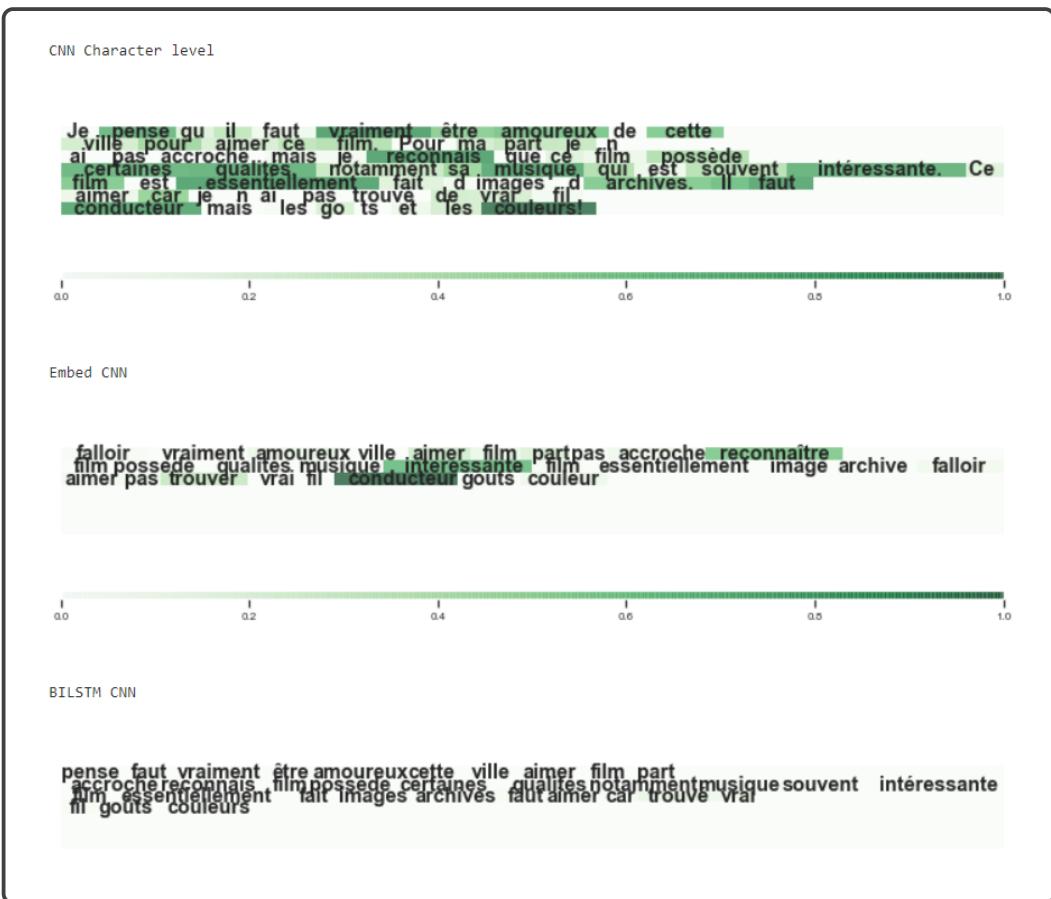


FIGURE 20 – Application de GradCAM pour la classe 1 sur l'exemple 13342

GradCAM Pour ce type d'exemple, il est parfois compliqué de comprendre la sortie du GradCAM car un mot peut être mis en avant pour les deux classes. A la fois pour la classe 0 et 1, les résultats pour Embedding CNN et le modèle à la maille caractère ne semblent pas avoir d'intérêt pour mettre en lumière le doute de ces deux modèles. Pour le BILSTM, c'est différent pour la classe 0 (Annexe 30) avec la mise en avant du *bi-gram* "aimer film" pour expliquer cet avis négatif.

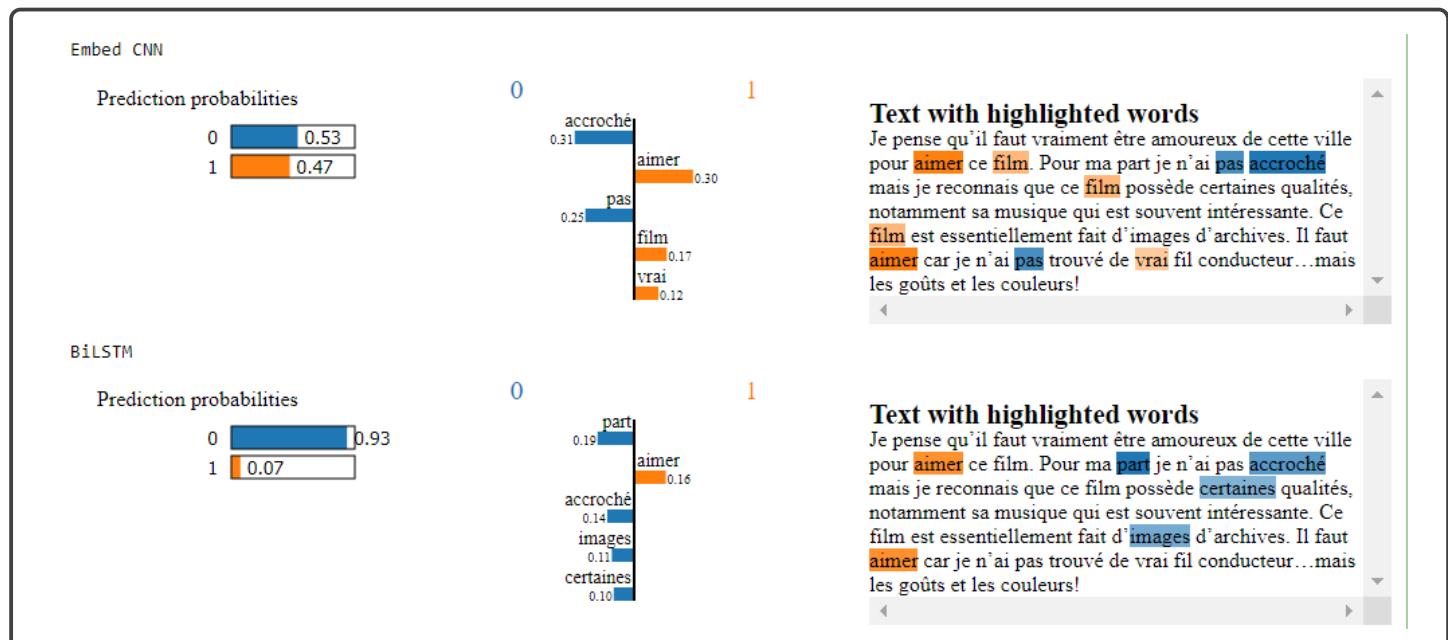


FIGURE 21 – Application de LIME sur l'exemple 13342

LIME Pour le modèle Embedding CNN, le LIME semble bien expliquer les doutes de la prédictions avec "pas" et "accroché" qui font pencher la probabilité du modèle vers 0; "aimer" et "film" qui font au contraire infléchir la probabilité vers 1. De plus les explications du BiLSTM, qui est le seul modèle à classifier correctement ce commentaire, ne sont pas concluant.

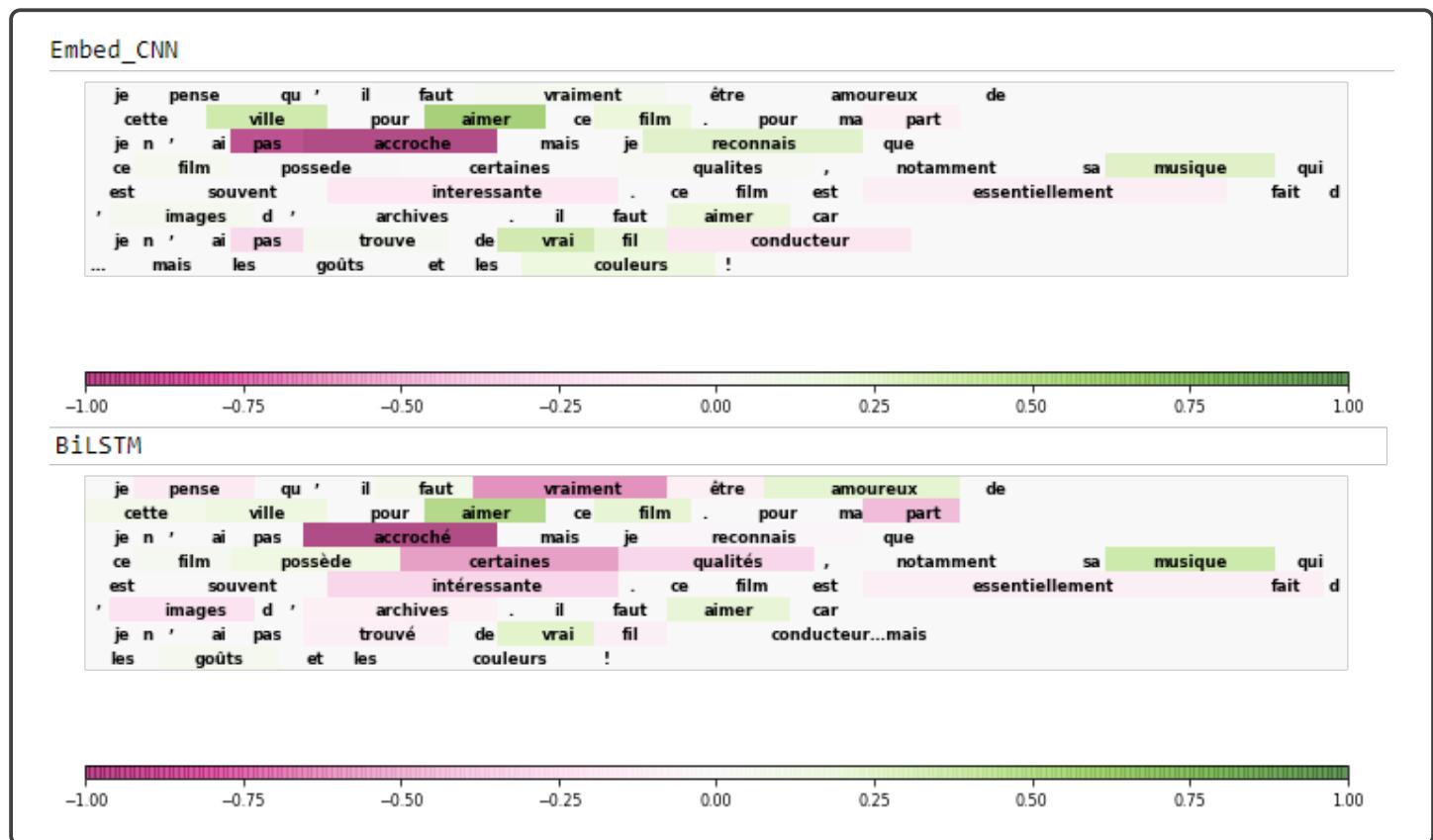


FIGURE 22 – Application de shap sur l'exemple 13342

SHAP Pour le modèle Embedding CNN, SHAP arrive à expliquer la prédiction de 0.5. Cependant, il ne marche ici pas très bien pour le modèle BiLSTM.

Remarques :

D'une manière générale, les trois méthodes d'interprétabilité sont intéressantes quand le modèle prédit correctement le commentaire. Même s'il est parfois possible de comprendre pourquoi le modèle est indécis (grâce à SHAP et au LIME notamment), il est souvent difficile comprendre pourquoi une prédiction n'est pas bonne. Dans ces cas-là, il est compliqué de savoir si le problème vient de la méthode d'interprétabilité ou du modèle de classification. Par exemple, lorsque des mots comme "Gérard Jugnot" apparaissent importants dans la prédiction du modèle, *a priori*, il est difficile de savoir si cela provient d'un problème de sur-apprentissage des modèles, ou bien des méthodes d'interprétabilité⁹.

9. *A priori*, pour le problème de "Gérard Jugnot", la probabilité estimée (sur le jeu d'entraînement) pour un commentaire contenant "Jugnot" d'être négatif est de 61.2%. Comme la probabilité de voir un commentaire négatif dans le jeu d'entraînement est 50%; difficile de penser que le problème de "Gérard Jugnot" est lié à du sur-apprentissage

3 Limites et améliorations

3.1 Limites computationnelles et logistiques

Ce projet a été développé en utilisant le *framework Pytorch de Python*.

L'entraînement des réseaux de neurones est très coûteux en ressource computationnelle. Il est recommandé de disposer d'une carte graphique (GPU) moderne afin de pouvoir les entraîner dans un délai raisonnable. Malheureusement, ce n'était pas le cas de tous les membres du groupe : ceci a pu provoquer à certains moments un manque d'efficacité dans la réalisation des tâches.

De plus, certains hyperparamètres ou choix sur l'entraînement (comme les algorithmes d'optimisation) n'ont pu être testés de manière approfondie. L'article de [Kim14] recommandait d'utiliser deux fois des algorithmes de validation croisée afin de contrôler l'erreur d'estimation et l'erreur inhérente à l'utilisation d'algorithme d'optimisation stochastique.

Ce problème de manque de ressources est aussi ressenti lors de l'utilisation des méthodes d'interprétabilité agnostiques. En effet, par exemple, pour LIME, il faut créer un grand nombre de réplications bruitées de l'observation considérée puis entraîner des modèles sur ce jeu de données. Ces opérations sont très longues. Ce problème est impactant également lors de la recherche des hyperparamètres optimaux : ces derniers étant le fruit d'une règle du pouce, il est nécessaire de relancer un grand nombre de fois ces modèles afin de les trouver. C'est pour cela que dans la section 2.2.1 les méthodes LIME et SHAP n'apparaissent pas, car itérer sur toutes les données de test aurait demandé plus d'une vingtaine d'heure de calcul par méthode (sans considérer la recherche des hyperparamètres, impossible sans fonction objective).

3.2 Limites inhérentes aux modèles

La méthode GradCAM est prometteuse. En effet, les figure 9 et 10 mettent en avant les mots ou les *bi-gram* les plus importants pour chaque modèle : ces explications sont cohérentes. La force du GradCAM est d'exploiter les couches de convolution des modèles, ce qui permet de mettre en avant des couples de mots significatifs. Contrairement à SHAP et à LIME qui fonctionnent par *substitution* oubliant ainsi le caractère dépendant d'une suite de mots qui constitue une phrase.

Néanmoins, les exemples atypiques qui ont été présentés précédemment ont mis en lumière certaines faiblesses pour la méthode GradCAM. Par exemple, lorsque les modèles hésitent (probabilité prédictive proche de 0.5), le GradCAM ne fournit pas de résultat concluant contrairement aux deux autres méthodes. Ce point est une grande faiblesse car les modèles d'interprétabilité sont très utiles lorsque le modèle propose des prédictions autour de 0.5 (dans le cas de la classification binaire).

Malgré tout, des pistes d'améliorations sont possibles. Par exemple, pour le modèle à la maille caractère, seule la première couche de convolution est exploitée. Il pourrait être envisageable d'utiliser une méthode de *déconvolution* pour pouvoir associer à nouveau la *feature map* artificielle de la dernière couche à l'intrant afin de ne pas à avoir à utiliser du *padding* massivement.

Autre exemple avec le modèle BiLSTM, pour éviter d'avoir une *feature map* vide comme dans la Figure 20, il serait préférable de *padder* les données après le passage de la couche LSTM et pas avant comme cela a été implémenté. En effet, en profitant du fait que l'entrant d'un LSTM peut ne pas être fixe, il aurait été possible de rentrer tous les commentaires qui contenaient moins de 67 mots (si plus de 67 mots, alors le commentaire est tronqué) et de *padder* si nécessaire à la sortie du LSTM (donc à l'entrée de la couche de convolution).

Enfin, avec le modèle Embedding CNN, il est possible d'aller plus loin en exploitant toute les *feature maps* artificielles¹⁰ qui sont générés en même temps grâce aux différents filtres. Dans l'application du GradCAM, celle qui a été utilisée est une agrégation des deux *features map* (car deux tailles différentes de filtre). Ainsi, il serait possible d'afficher pour chaque filtre ce type d'explication (Figure 11) et obtenir un pouvoir explicatif plus ingénieux.

10. Le modèle Embedding CNN est défini à l'aide de deux couches de convolutions qui sont "empilées"

Conclusion

Le but de ce projet était de construire un outil d’interprétabilité pour les équipes métiers. Il existe des méthodes agnostiques comme SHAP ou LIME mais chacune présente des limites. Par exemple pour LIME, il est nécessaire d’ajuster les hyperparamètres d’un modèle pour construire des interprétations. Pour autant, il n’existe pas de fonctions objectif permettant de quantifier l’interprétabilité d’une explication et donc ces hyperparamètres ne sont pas le fruit d’un problème d’optimisation et ne peuvent être trouvés automatiquement avec des méthodes numériques. Des règles du pouces sont donc utilisées pour trouver ces paramètres. D’autres critiques sont faites à ces modèles d’interprétabilité : d’après l’article [Kum+20], la méthode SHAP ne serait pas une méthode efficace pour chercher expliquer la sortie d’un modèle.

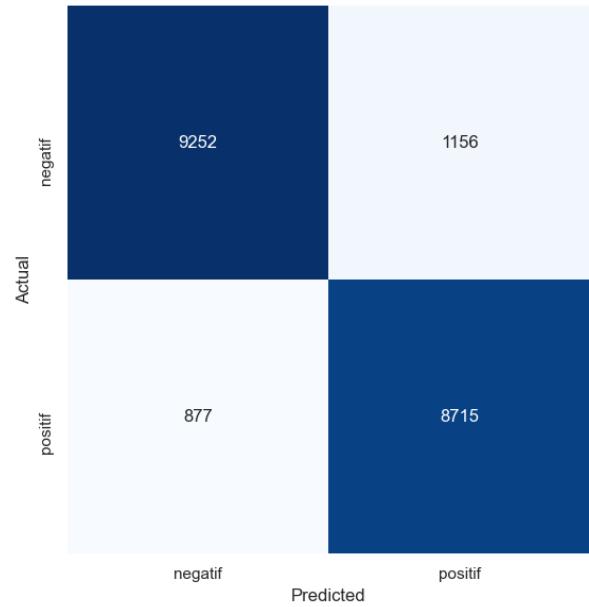
La méthode GradCam permet de prendre en compte la structure des réseaux de neurones contenant des couches convolutionnelles. Ces méthodes semblent donner des résultats plus concluants que LIME ou SHAP sauf dans le cas où le modèle est indécis. De plus, cette méthode ne nécessite pas d’utilisation de surmodèles : il n’est pas nécessaire d’apprendre un nouveau modèle et donc d’ajouter une source supplémentaire d’approximation et d’erreur.

Plusieurs pistes d’améliorations ont été identifiées pour cette méthode : utilisation de modèles de déconvolution, autres stratégies de convolution afin d’améliorer ses performances.

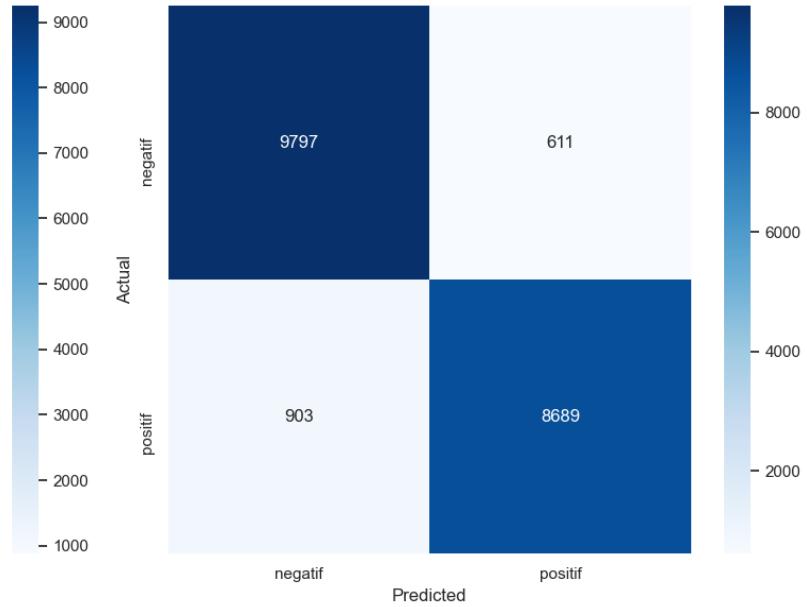
Ces méthodes ont été utilisées sur du texte ou des images. Cependant, les réseaux de neurones convolutionnels sont également utilisés pour la prédiction pour des séries temporelles. Un reproche pouvant être fait à certaines méthodes de prédiction en série temporelle est le manque d’explication. En effet, ces explications peuvent être fondamentales si ces prédictions sont utilisées par le décideur public par exemple. On pourrait utiliser ces méthodes d’interprétabilité par exemple pour avoir des éléments explicatifs lors de prédictions de séries temporelles comme des prédictions démographiques.

Annexes

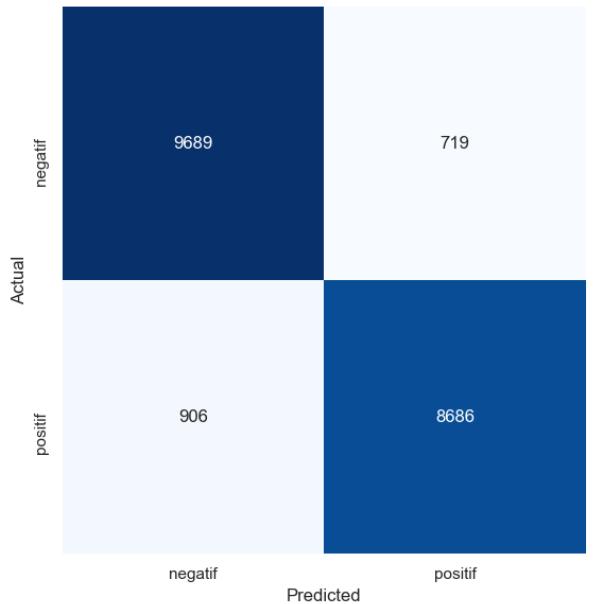
FIGURE 23 – Matrice de confusion



(a) Embedding CNN

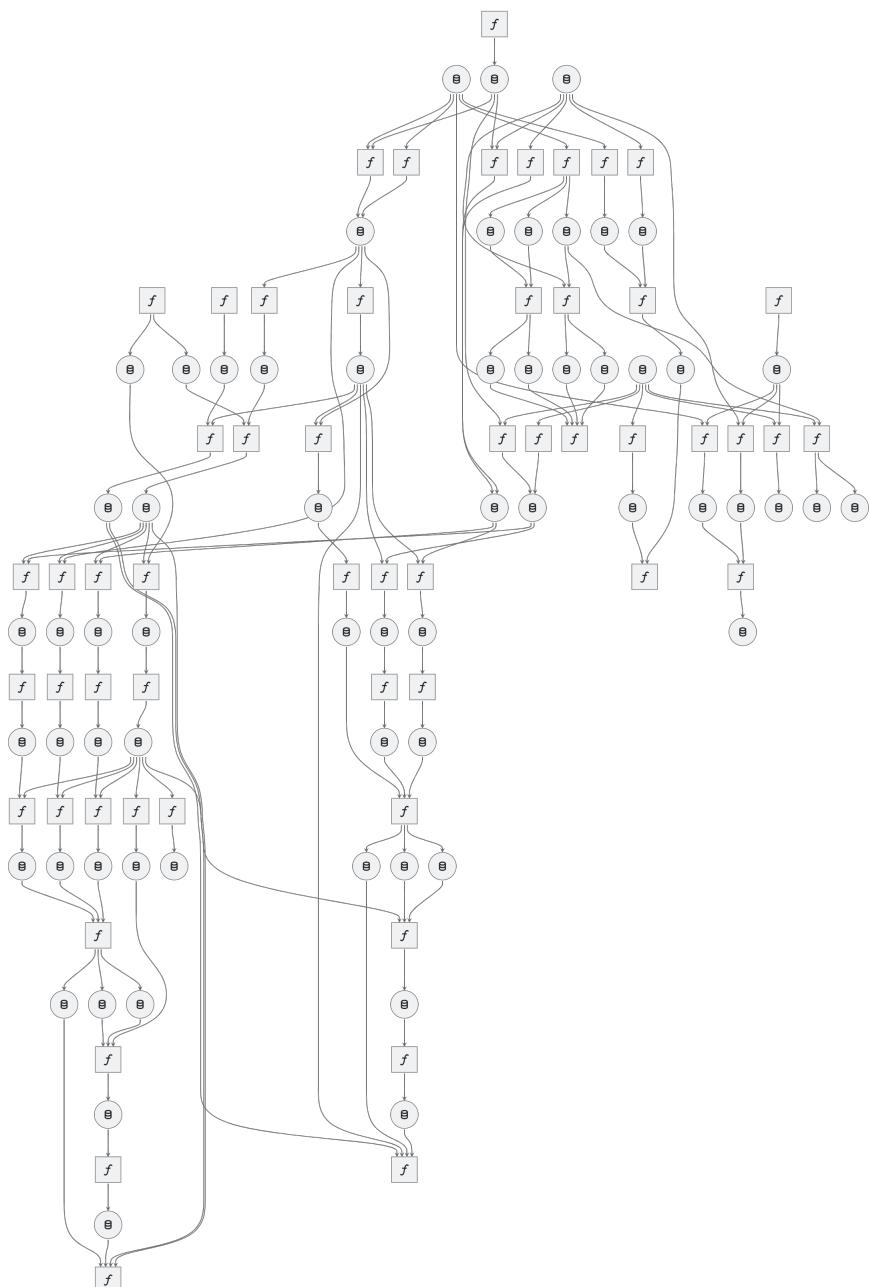


(b) Caractères CNN



(c) BILSTM CNN

FIGURE 24 – Visualisation de l'architecture du projet permise grâce à Kedro



Code Listing 1 – Implémentation PyTorch du modèle BiLSTM avant le GradCAM

```
import torch.nn as nn
import torch

class BilstmCnn(nn.Module):
    def __init__(self, embedding_matrix, sentence_size, input_dim, hidden_dim, layer_dim,
                 output_dim, feature_size
                 , kernel_size, dropout_rate, padded):
        super(BilstmCnn, self).__init__()

        self.mp_kernel_size = 2
        self.linear_dim = int(2*hidden_dim * (sentence_size-kernel_size)/2)
        self.embedding = nn.Embedding.from_pretrained(nn.Parameter(torch.tensor(embedding_matrix,
                                                                           dtype=torch.float32)), padding_idx=0)
        self.LSTM = nn.LSTM(input_size=input_dim, hidden_size=hidden_dim, bidirectional=True,
                            num_layers=layer_dim, dropout=dropout_rate, bias=True)

        self.convLayer = nn.Sequential(
            nn.Conv1d(in_channels=2*hidden_dim, out_channels=feature_size, kernel_size=kernel_size,
                     bias=True),
            nn.BatchNorm1d(2*hidden_dim),
            nn.ReLU())

        self.maxpool = nn.MaxPool1d(kernel_size=self.mp_kernel_size)

        self.fc = nn.Linear(self.linear_dim, output_dim)

    def forward(self, x):
        out = self.embedding(x)

        out = out.permute(1,0,2)

        out, (hn, cn) = self.LSTM(out)

        out = out.permute(1,2,0)

        out = self.convLayer(out)

        out = self.maxpool(out)

        out = out.reshape(out.size(0), -1)

        out = self.fc(out)

        return out
```

Code Listing 2 – Implémentation PyTorch du modèle BiLSTM après le GradCAM

```

import torch.nn as nn
import torch
from deep_nlp.grad_cam.model import GradCamBaseModel

class BilstmCnn(GradCamBaseModel):
    def __init__(self, embedding_matrix, sentence_size, input_dim, hidden_dim, layer_dim,
                 output_dim, feature_size
                 , kernel_size, dropout_rate, padded):
        super(BilstmCnn, self).__init__()

        self.padded = padded
        self.kernel_size = kernel_size
        self.mp_kernel_size = 2

        if not self.padded:
            self.linear_dim = int(2 * hidden_dim * (sentence_size - kernel_size) / 2)
        else:
            self.linear_dim = int(2 * hidden_dim * (sentence_size - 1) / 2)

        self.embedding = nn.Embedding.from_pretrained(nn.Parameter(torch.tensor(embedding_matrix,
                                                                           dtype=torch.float32))
                                                      , padding_idx=0)
        self.LSTM = nn.LSTM(input_size=input_dim, hidden_size=hidden_dim, bidirectional=True,
                            num_layers=layer_dim, dropout=dropout_rate, bias=True)

        self.convLayer = nn.Sequential(
            nn.Conv1d(in_channels=2 * hidden_dim, out_channels=feature_size, kernel_size=kernel_size,
                     bias=True),
            nn.BatchNorm1d(2 * hidden_dim),
            nn.ReLU())

        self.maxpool = nn.MaxPool1d(kernel_size=self.mp_kernel_size)

        self.fc = nn.Linear(self.linear_dim, output_dim)

        self.softmax = nn.LogSoftmax(dim=1)

    # Fill up pipelines
    self.before_conv.add_module("conv", self.convLayer)
    self.pool.add_module("maxpool", self.maxpool)
    self.after_conv.add_module("fc", self.fc)
    self.after_conv.add_module("sm", self.softmax)

    def get_activations(self, x):
        # Documentation said to !!!
        # Each forward step, reset gradient list to only get the one from the actual run (=from this
        # forward step)
        self.reset_gradient_list()

        if self.padded:
            x = nn.ZeroPad2d((0, self.kernel_size - 1, 0, 0))(x)

        x = self.embedding(x)
        x = x.permute(1, 0, 2)
        x, _ = self.LSTM(x)
        x = x.permute(1, 2, 0)
        x = self.before_conv(x)
        return x

    def forward(self, x):
        x = self.get_activations(x)

        if x.requires_grad:
            h = self.register_hook(x)

        x = self.pool(x)
        x = x.reshape(x.size(0), -1)
        x = self.after_conv(x)

        return x

```

FIGURE 25 – Fréquence des mots les plus importants pour la classe 0, pour les commentaires classifiés négatifs

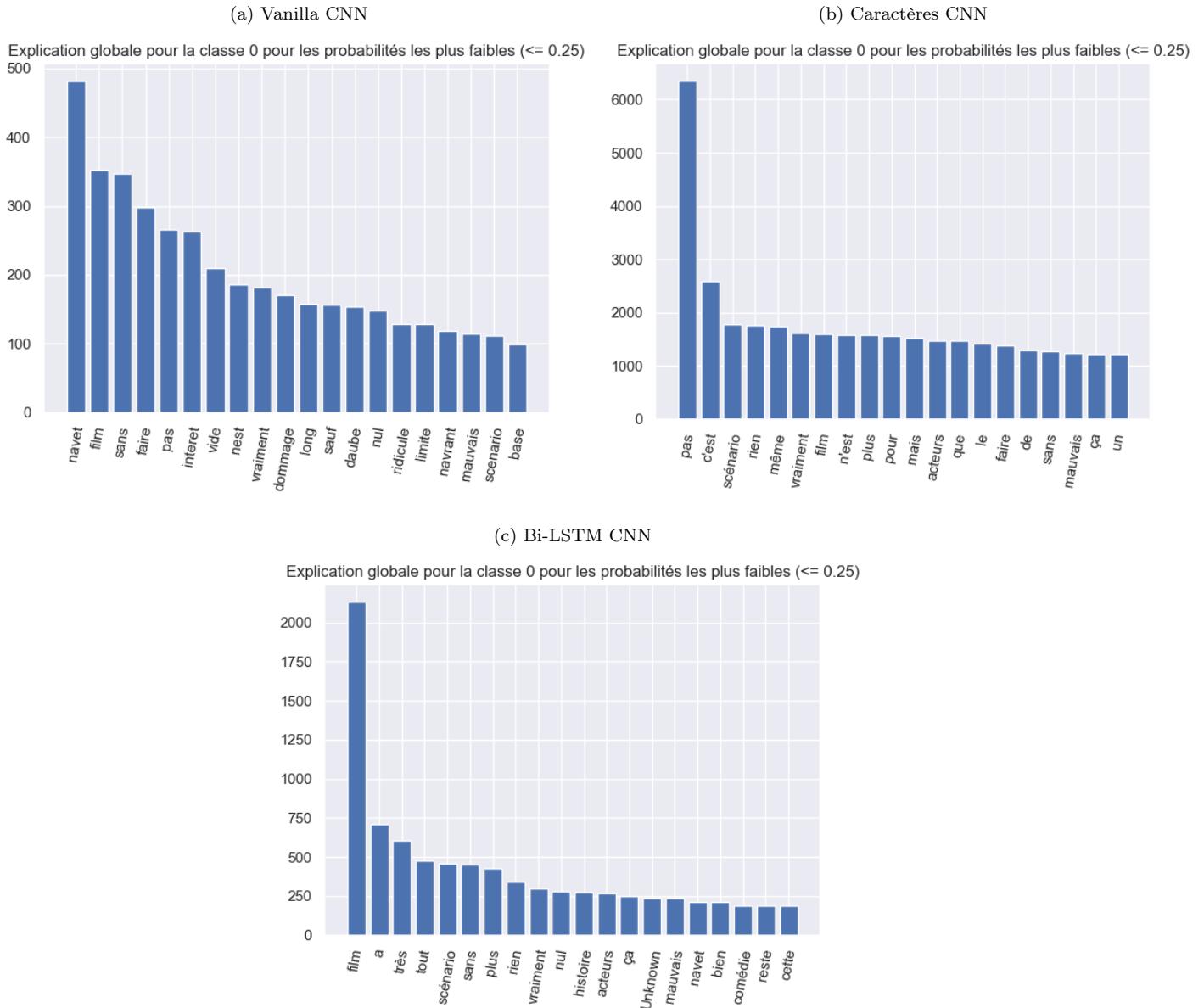
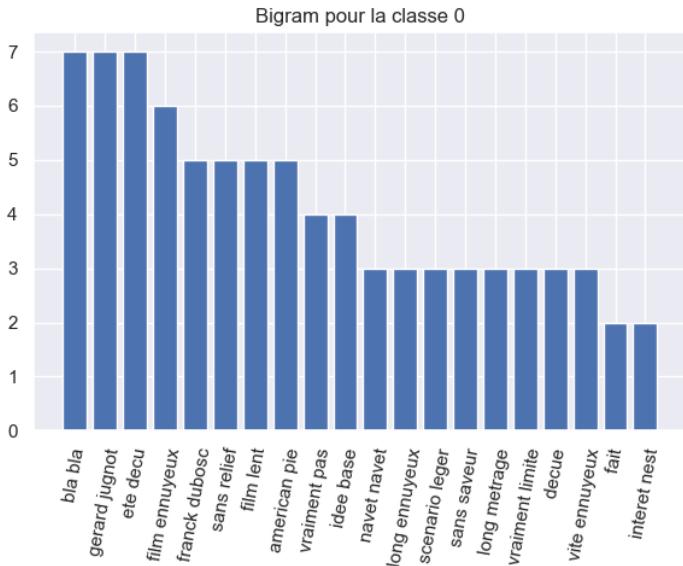
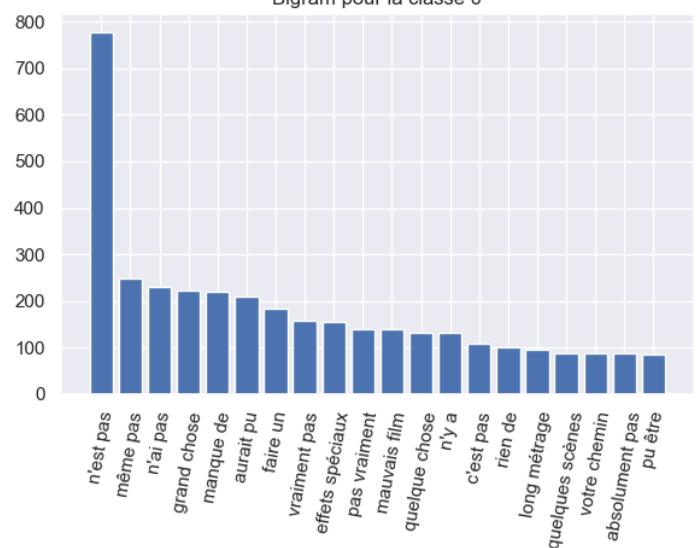


FIGURE 26 – Fréquence des bi-grams les plus importants pour la classe 0, pour les commentaires classifiés négatifs

(a) Vanilla CNN



(b) Caractères CNN



(c) Bi-LSTM CNN

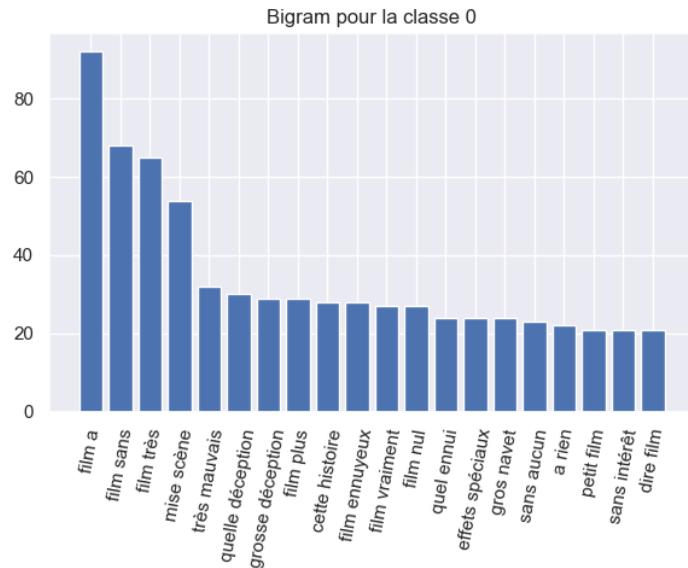


FIGURE 27 – Application de GradCAM pour la classe 0 sur l'exemple 516

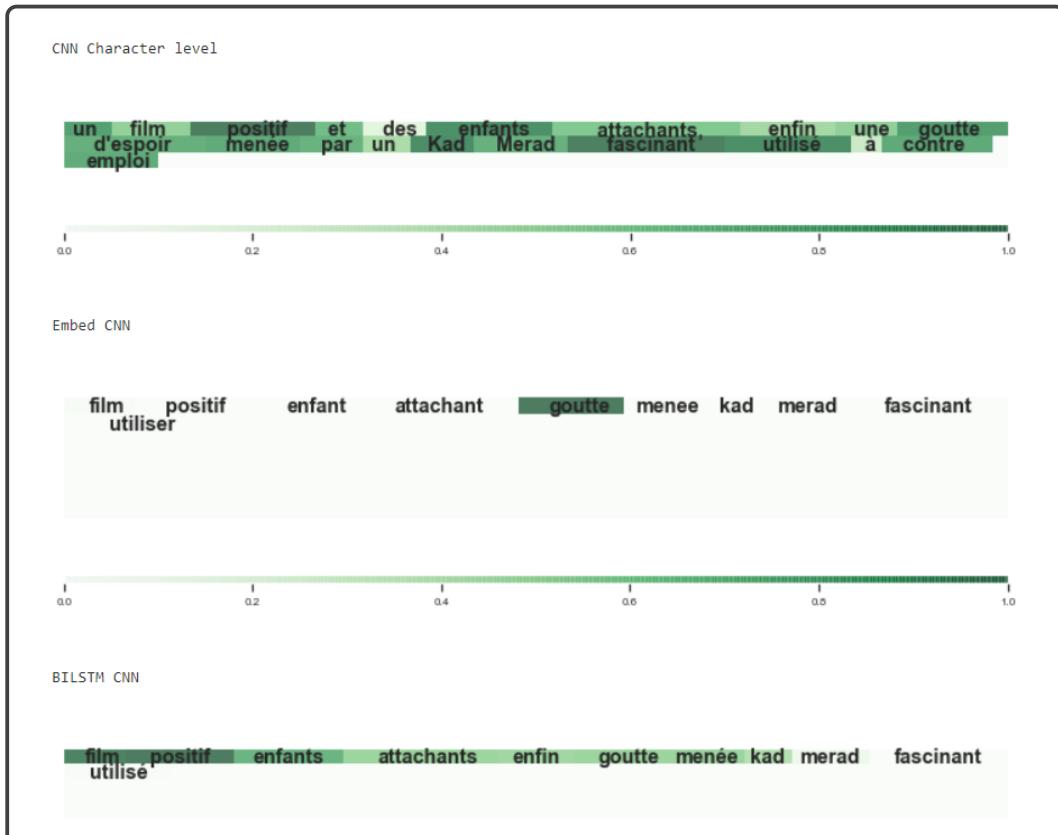


FIGURE 28 – Application de GradCAM pour la classe 0 sur l'exemple 8619

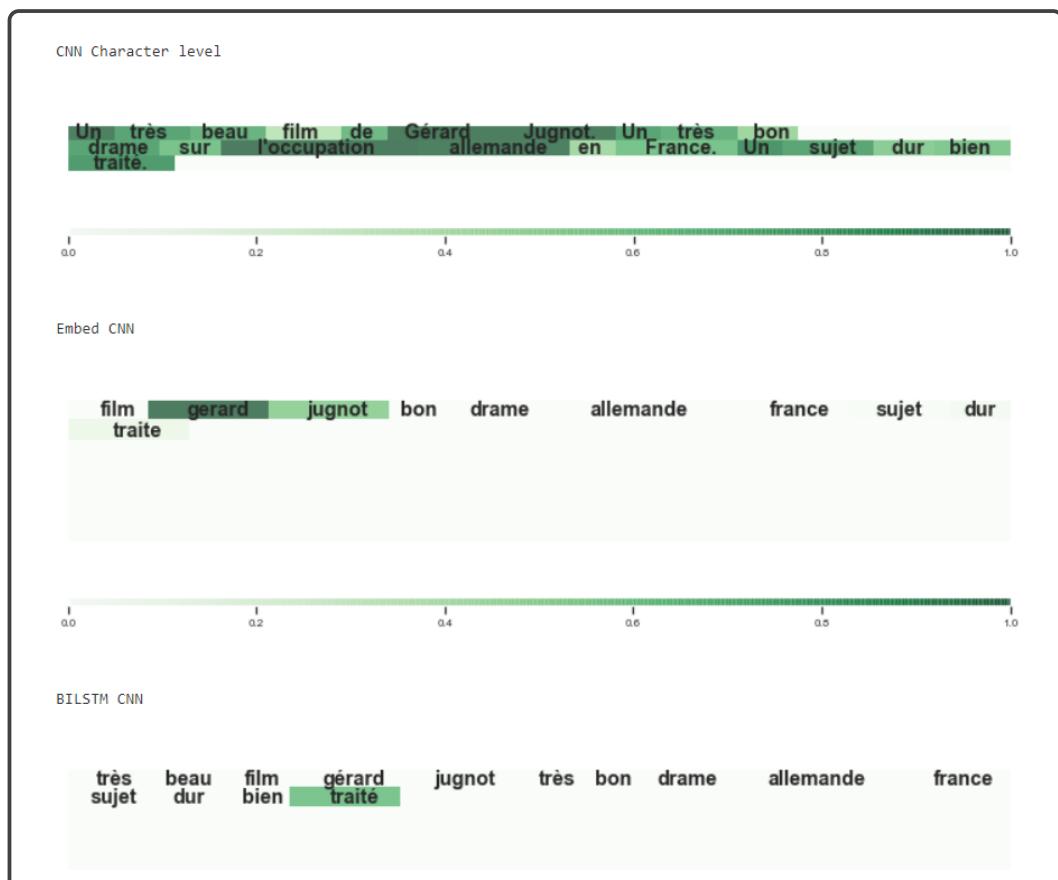


FIGURE 29 – Application de GradCAM pour la classe 1 sur l'exemple 1287

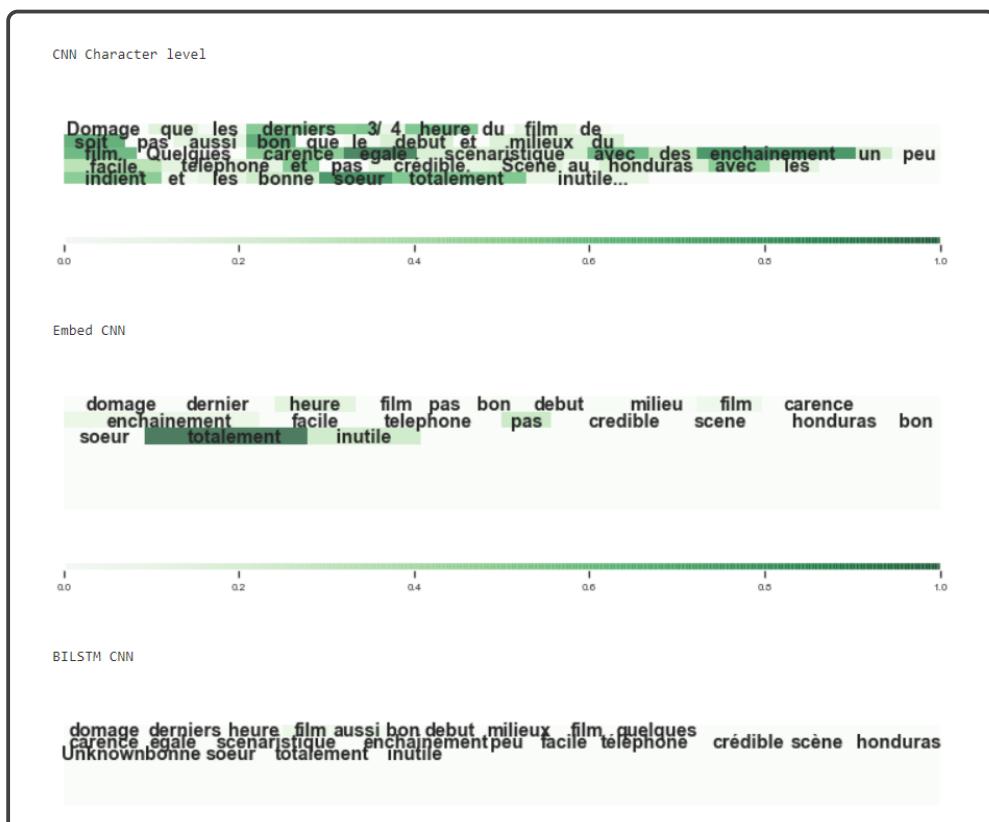
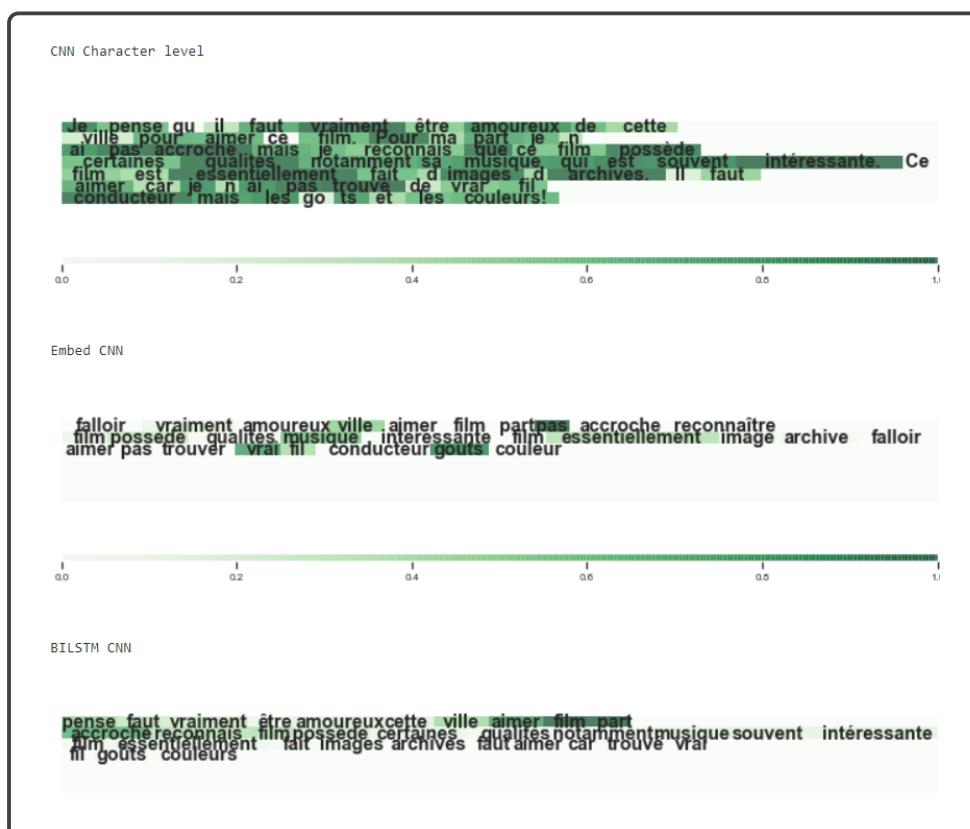


FIGURE 30 – Application de GradCAM pour la classe 0 sur l'exemple 13342



Bibliographie

- [1] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E HINTON. "ImageNet Classification with Deep Convolutional Neural Networks". In : *Advances in Neural Information Processing Systems*. Sous la dir. de F. PEREIRA et al. T. 25. Curran Associates, Inc., 2012. URL : <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.
- [2] Tomas MIKOLOV et al. "Distributed Representations of Words and Phrases and their Compositionality". In : (2012). URL : <https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf>.
- [3] Yoon KIM. "Convolutional Neural Networks for Sentence Classification". In : *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar : Association for Computational Linguistics, oct. 2014, p. 1746-1751. DOI : 10.3115/v1/D14-1181. URL : <https://www.aclweb.org/anthology/D14-1181>.
- [4] Karen SIMONYAN et Andrew ZISSERMAN. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In : *CoRR* abs/1409.1556 (2014). URL : <http://arxiv.org/abs/1409.1556>.
- [5] Xiang ZHANG, Junbo ZHAO et Yann LECUN. "Character-level Convolutional Networks for Text Classification". In : *Advances in Neural Information Processing Systems*. Sous la dir. de C. CORTES et al. T. 28. Curran Associates, Inc., 2015. URL : <https://proceedings.neurips.cc/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf>.
- [6] Bolei ZHOU et al. "Learning Deep Features for Discriminative Localization." In : *CoRR* abs/1512.04150 (2015). URL : <http://dblp.uni-trier.de/db/journals/corr/corr1512.html#ZhouKLOT15>.
- [7] Piotr BOJANOWSKI et al. "Enriching Word Vectors with Subword Information". In : <https://arxiv.org/pdf/1607.04606.pdf> (2016).
- [8] Ian J. GOODFELLOW, Yoshua BENGIO et Aaron COURVILLE. *Deep Learning*. <http://www.deeplearningbook.org>. Cambridge, MA, USA : MIT Press, 2016.
- [9] Marco Tulio RIBEIRO, Sameer SINGH et Carlos GUESTRIN. ""Why Should I Trust You ?" : Explaining the Predictions of Any Classifier". In : (2016), p. 1135-1144.
- [10] Peng ZHOU et al. "Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling". In : *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics : Technical Papers*. Osaka, Japan : The COLING 2016 Organizing Committee, déc. 2016, p. 3485-3495. URL : <https://www.aclweb.org/anthology/C16-1329>.
- [11] Ramprasaath R. SELVARAJU et al. "Grad-CAM : Visual Explanations from Deep Networks via Gradient-Based Localization." In : *ICCV*. IEEE Computer Society, 2017, p. 618-626. ISBN : 978-1-5386-1032-9. URL : <http://dblp.uni-trier.de/db/conf/iccv/iccv2017.html#SelvarajuCDVPB17>.
- [12] Jacob DEVLIN et al. "BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding". In : *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota : Association for Computational Linguistics, juin 2019, p. 4171-4186. DOI : 10.18653/v1/N19-1423. URL : <https://www.aclweb.org/anthology/N19-1423>.
- [13] Aston ZHANG et al. *Dive into Deep Learning*. <http://www.d2l.ai>. 2019.
- [14] I. Elizabeth KUMAR et al. "Problems with Shapley-value-based explanations as feature importance measures". In : *CoRR* abs/2002.11097 (2020). arXiv : 2002.11097. URL : <https://arxiv.org/abs/2002.11097>.