UNIVERSITE DE PARIS
UFR MATHEMATIQUES ET INFORMATIQUE

Deep Learning pour l'analyse de texte

Master 1 Vision Machine Intelligente

Seyedeh Neguin NAVIDI – Luca BRESOLIN

Encadré par Nicole VINCENT ET BAPTISTE BOHET

Année universitaire 2020 – 2021

Table Des Matières

Table des matières

1. Introduction

Remarquer qu'un livre n'a pas été écrit dans notre langue initialement n'est pas forcément quelque chose d'évident pour tout le monde, néanmoins des lecteurs assidus, des bilingues, des polyglottes ou des linguistes ont souvent remarqué être capable de deviner cette partie de la nature du texte sans information contextuelle. C'est de ce constat qu'étaient partis nos professeurs encadrants, en imaginant que si un humain ayant certaines connaissances est capable de distinguer un texte « natif » d'un texte traduit depuis une autre langue, alors l'ordinateur pourrait bien lui aussi être capable de réaliser cette tâche à l'aide notamment de techniques issues de l'intelligence artificielle.

Il nous fallait alors nous demander : Comment notre cerveau parvient ils à différencier les textes sur cette base ? Quels éléments, quels mots, quel type de vocabulaire ou de construction syntaxique sont les plus révélateurs ? Est-ce la richesse lexicale ou bien le style d'écriture qui provoque en nous cette intuition ? Les différents facteurs intervenant dans notre observation sont parfois flous et difficile à expliciter, nous pouvions entendre dire : « Je sais que c'est un original, mais je ne saurai pas dire pourquoi. »

Notre objectif dans ce projet était donc de trouver une manière d'automatiser cette tâche et de créer un programme capable de discriminer le plus précisément possible un livre qui a été écrit dès le départ en français d'un texte d'abord écrit dans une autre langue puis traduit en français, nous nous cantonnerons ici à des textes originellement publiés en anglais.

Pour accomplir cette tâche, nous nous sommes inspirés de ce qui a été fait par les experts du domaine, mais nous avons également essayer d'implémenter des méthodes conseillées par nos tuteurs.

2. Etat de l'art

Notre travail s'inscrit dans le cadre du traitement automatique du langage naturel qui est un champ multidisciplinaire dans lequel interviennent notamment la linguistique et l'apprentissage machine. Nous nous intéresserons ici plus particulièrement à la classification de textes puisque notre sujet consiste à diviser les textes en deux catégories : traduits et natifs.

Selon cet article (Qian Li, 2022) recensant les différentes approches majeures en classification textuelle, il y a deux méthodes classiques qui sont actuellement utilisées : la méthode dite traditionnelle et une autre méthode utilisant l'apprentissage profond.

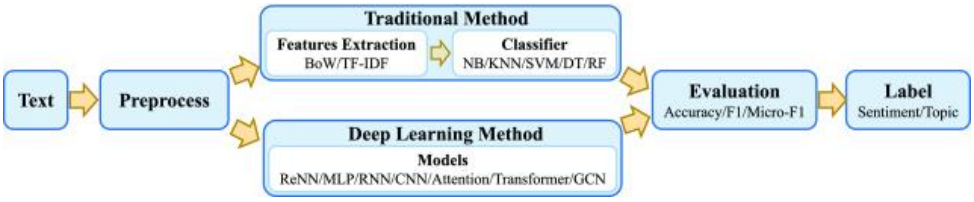


Figure 1 - Deux méthodes classiques de classification textuelle (issue de Qian Li, 2022)

La première méthode comporte deux étapes principales : l'extraction de caractéristique, très souvent effectuée à l'aide de méthodes inspirées des représentations Sac de mots (BagOfWords) ou de TF-IDF qui modifient toutes la forme des données, ensuite on donne ces données à des modèles d'apprentissage classiques, comme par exemple ceux établis sur le principe des plus proches voisins ou ceux bâtis sur le théorème de Bayes (ces classifieurs bayésiens sont largement répandus, sans doute grâce leur simplicité, leur rapidité et leur capacité à apprendre sur des données peu nombreuses)

La deuxième méthode se base entièrement sur un réseau de neurones profonds que ce soit en apprentissage supervisé ou non. Ces derniers sont très populaires depuis quelques années et ont prouvés leur grande efficacité. Leurs résultats sont souvent supérieurs à ce que l'on peut obtenir avec des méthodes traditionnelles, cependant leur bon fonctionnement est dépendant de données présentes en grand nombre.

Sur ce sujet précis, quelques études sont disponibles, comme cette publication (Cyril Goutte, 2009), les résultats obtenus étaient satisfaisants quoiqu'améliorables.

Contribution

Dans l'objectif d'obtenir les meilleurs résultats possibles, nous avons exploré différentes approches au cours du semestre. Nous pouvons les résumer en trois principales catégories : les méthodes « naïves », les méthodes visuelles et les méthodes vectorielles.

Les données auxquelles nous avions accès consiste en un total de plus de 2000 textes, parmi lesquels 1704 étaient exploitables (dont 964 textes natifs et 740 textes traduits). Ces données textuelles pèsent environ 1Go et chaque livre est de taille et d'origine variable, sans exclure toutefois un possible biais d'échantillonnage, le corpus contenant tout de même de multiples œuvres d'un même auteur et des œuvres appartenant souvent à des genres spécifiques : beaucoup de livre contemporain, beaucoup de polars. Néanmoins, nous pensons que la taille de notre échantillon permet de limiter le plus possible un apprentissage trop spécifique qui ne parviendrait pas à généraliser assez bien.

3. Méthodes étudiées

3.1 Méthodes « naïves »

Ce premier type d'approche suppose que les textes traduits et les textes originaux possèdent des caractéristiques visibles qui les distinguent. Il convient alors de repérer ces caractéristiques, de les extraire, de les numériser si nécessaire et enfin de les fournir à un modèle de classification qui s'efforcera de distinguer les deux types de texte sur la seule base de ces dernières.

Parmi les caractéristiques qui nous semblaient en moyenne différentes entre un texte natif et un texte traduit, nous avions trouvé :

- La richesse lexicale (e.g proportion de mots rares, proportion de mots répétés)
- La fréquence d'anglicismes (e.g « faire sens », « virtuellement » à la place de « pratiquement », etc)
- La longueur moyenne des phrases

- La longueur moyenne des mots
- Des proportions différentes en termes de catégorie grammaticale (i.e nature ou partie du discours) des mots

Une fois ces différentes caractéristiques repérées, nous devions alors les extraire et vérifier si elles étaient distribuées différemment entre les textes traduits et les textes originaux. Nous pouvons voir sur la Figure 1 la distribution de différents indicateurs de la richesse lexicale des textes, nous observons bien dans cet exemple que la variété (définie comme le rapport entre le nombre de mots différents sur le nombre de mots total d'un texte) possède une moyenne et un écart-type différents pour chaque classe, ce qui nous indique qu'à priori cette caractéristique est intéressante pour distinguer les textes.

Après la sélection des meilleurs indicateurs, nous les rassemblons sous forme de liste et chaque texte se voit attribuer une liste de caractéristiques lui correspondant. Les textes ainsi transformés sont fournis à un modèle d'apprentissage machine effectuant une classification à 2 classes, comme MultiNomialNB (classificateur bayésien naïf).

3.2 Méthode vectorielle

Tout comme l'approche précédente, nous souhaiterons ici aussi extraire de chaque texte certaines caractéristiques afin de simplifier le traitement de ces données. Nous avons opté pour une représentation vectorielle de chaque texte, cette représentation obtenue par la méthode CountVectorizer de la bibliothèque sci-kit learn permet de transformer un corpus de textes en une liste de vecteurs où chaque texte a été transformée en un vecteur équivalent à l'aide des informations contenues dans le corpus.

	big	count	create	dataset	differnt	features	hello	is	my	name	notebook	of	python	this	to	try	trying	vectorizer	words
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
3	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	0	0	1
4	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0

Figure 2 - Vecteurs résultant de la transformation du corpus suivant : textes = ['Hello my name is james','james this is my python notebook','james trying to create a big dataset','james of words to try differnt','features of count vectorizer'] par la fonction CountVectorizer

Plus précisément, comme nous pouvons le voir sur la figure ci-dessus, chaque mot différent présent dans le texte devient une dimension et chaque texte est transformé en données numériques vectorielles dont chaque valeur correspond au nombre d'occurrences du « mot-dimension » dans ce dernier.

Nous avons parlé ici de mots, mais nous pouvons faire pareil avec des couples de mots, des séquences de n mots (word gram), des n_grams (i.e séquence de n caractères contigus dans le texte). Parmi les autres paramètres disponibles, nous pouvez choisir le nombre maximum de mots choisis (ergo la dimensionnalité maximale), d'ignorer certains mots (e.g selon une liste prédéfinie, pas assez ou trop présent selon des seuils définis par l'utilisateur).

Deuxièmement, la forme sous laquelle nous insérons nos textes dans la méthode de vectorisation est elle aussi importante, nous pouvons soit donner les textes tels quels, soit pour multiplier le nombre d'échantillons, il nous est possible de donner des segments de texte d'une taille arbitraire (nous la noterons K).

Enfin avec les textes ainsi transformés, nous les fournissons encore une fois à un modèle de classification qu'il nous faudra là aussi choisir, notamment parmi les modèles classiques tels que les machine à vecteurs de support, les classificateurs bayésiens naïf, les modèles des K plus proches voisins et autres.

Comme nous venons de l'évoquer, de nombreux choix et paramètres sont disponibles et il nous fallait trancher, nous avons alors exploré le plus de possibilités pour ensuite les comparer afin de prendre une décision éclairée.

3.3 Méthodes basées sur les images

Nous avons implémenté cette méthode proposée par nos tuteurs car beaucoup d'outils sont disponibles pour travailler avec des images notamment des réseaux de neurones profonds performants.

Dans ce qui suit, nous allons voir les méthodes utilisées pour créer les images et quelques bases nécessaires pour cette section, après cela, nous aborderons les méthodes de création d'images, dans la troisième partie, nous verrons les parties d'apprentissage automatique et enfin nous discuterons des résultats pratiques.

3.3.1 Création des images

Travailler avec des données sous forme d'image est différent de travailler avec des données textuelles. Dans le texte, nous avons une séquence de caractères pouvant former un mot et un texte, mais dans l'image, nous n'avons que des entiers compris entre 0 et 255. Cela signifie que nous ne pouvons pas utiliser directement nos données textuelles dans les images. Pour transférer les informations du texte à nos images, nous avons décidé de choisir certaines caractéristiques du texte et de les traduire en un entier compris entre 0 et 255. Dans la section suivante, nous allons voir les méthodes que nous avons choisies pour créer les images en détail.

3.3.2 Les trois types d'images RVB

Comme nous l'avons dit dans la dernière partie, nous devons trouver des méthodes pour traduire les informations dont nous disposons sur les textes en image. Nous pouvons avoir de nombreuses caractéristiques comme le nombre de mots pour chaque texte. Pour les images en RVB, nous avons trois couches pour maitre les caractéristiques cela veut dire que nous pouvons choisir trois caractéristiques différentes pour créer les images en RVB.

En discutant avec nos encadrants, nous avons choisi plusieurs caractéristiques avec lesquelles nous avons créé 3 types d'images en couleur, trois couches Rouge, Vert et Bleu, et ---- types d'images en une couche, niveaux de gris. Nous allons les expliquer une par une.

Premier type d'images

Pour ce type d'images nous avons choisi de couper les textes en caractères. La première couche de ce type d'image consiste des caractères codés. Pour coder les caractères, une optionne était d'utiliser le code ascii, mais nous avons décidé d'utiliser notre propre codage pour les caractères, vous pouvez voir les valeurs qu'on a choisi sur le tableau 1.

Pour la deuxième couche, la couche verte, nous avons utilisé la partie du discours[1]. Pour étiqueter la partie du discours de chaque mot que nous avons utilisé un outil qui s'appelle « TreeTagger ». Le TreeTagger est un outil pour annoter du texte avec des

[1] *En anglais : part of speech*

informations sur la partie du discours et le lemme. Il a été développé par Helmut Schmid dans le cadre du projet TC de l'Institut de linguistique computationnelle de l'Université de Stuttgart. Le TreeTagger a été utilisé avec succès pour baliser l'allemand, l'anglais, le français, l'italien, le danois, le suédois, le norvégien, le néerlandais, l'espagnol, le bulgare, le russe, le portugais, le galicien, le grec, le chinois, le swahili, le slovaque, le slovène, le latin, l'estonien, le polonais, persan, roumain, tchèque, copte et ancien français et est adaptable à d'autres langues si un lexique et un corpus d'apprentissage étiqueté manuellement sont disponibles. Il utilise un arbre de décision binaire pour trouver la partie du discours de chaque mot. (Schmid, 1994) . TreeTagger en entrée prend un morceau du texte et en sortie nous aurons une liste qui contient le mot comme la première valeur et sa POS comme la deuxième valeur et le lemme du mot comme troisième valeur. Un exemple de la sortie de cette partie est présenté sur la figure 1.

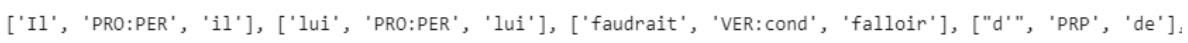


Figure 3 : un exemple de la sortie de TreeTagger

TreeTagger peut reconnaitre 34 catégories de partie du discours qui sont présenté sur le tableau 2. Encore une fois nous ne pouvons pas utiliser directement ces valeurs et on doit nous devons attribuer une valeur numérique ente 0 et 255 à chacun d'eux. Ces valeurs aussi sont présenté sur le tableau 2.

Pour que les caractères correspondants à un mots soient la même valeur de la partie de discours nous répétons cette valeur pour chaque caractère de mot correspondants. Cela nous laisse de supprimer les espaces entre les mots car un mot est reconnaissable par les valeurs de la deuxième couche.

Pour la troisième couche nous avons décidé d'utiliser une représentation de la fréquence des mots. Nous savons que la fréquence d'un mot comme 'de' et beaucoup plus grande que 255. Alors, une solution pour représenter la fréquence des mots entre 0 et 255 et de normaliser les fréquences mais comme ça on va perdre la fréquence des mot rares, comme les hapax. Un hapax est un mot ou forme qu'on ne rencontre qu'une fois dans un corpus donné. En linguistique, le terme hapax a qualifié les mots (ou expressions) que les traducteurs ne pouvaient traduire de façon certaine, car il n'en existait qu'une seule occurrence dans la littérature alors nous savons que nous devons garder ces mots. A cause de ces raisons nous avons associé les valeurs entre 0 et 127 aux 127 mots qui

sont les moins fréquents et les valeurs entre 129 et 255 aux 127 mots qui sont les plus fréquents et tous ce qu'il reste à 128. Pour cette couche aussi comme la deuxième couche, nous répétons la fréquence des mots correspondant à chaque mot pour tous ses caractères.

En utilisant ces trois couches, nous avons créé nos images en couleurs. Vous pouvez voir quelques exemples sur la figure 2 et 3.

Après avoir calculer les valeurs pour chaque couche, nous coupons les valeurs en 224*224 éléments, ici 224*224 caractères, et nous les enregistrons sous forme les fichiers '.PNG'. Avec cette méthode nous obtenons environ 10.000 images pour chaque catégorie.

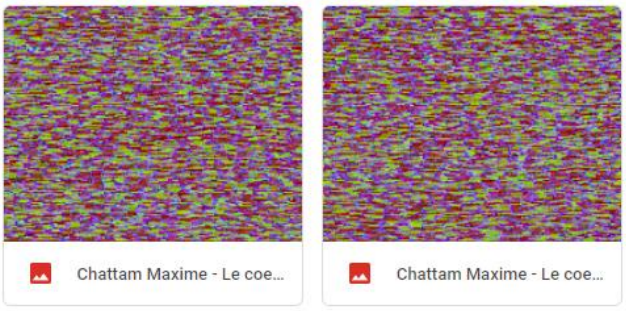


Figure 4: deux exemples des images d'un texte original- RVB

Figure 5: deux exemples des images d'un texte traduit- RVB

3.3.3 Les images en niveaux de gris

Après avoir testé les images en RGB, nous voulons comprendre pourquoi nos images ne sont pas assez représentatives. Alors nous avons décidé de voir ce qui se passe si nous prenons une seule caractéristique. Cela nous aide de savoir si les caractéristiques que nous avons utilisées sont bien choisit ou pas. Pour ces raisons nous avons créé les images en niveaux de gris, chaque type d'images contient une des caractéristiques utilisées pour créer les images en couleurs.

- Les caractères codés

Pour ce type d'images nous avons pris seulement la première couche des images de premier type, qui contient les caractères codés. Pour le premier type nous n'avons pas

pris en compte les caractères vides ou les espaces mais ici si nous ne les mettons pas, les images n'auront pas de sens alors nous avons remis les espaces. Vous pouvez voir quelques exemples de ce type d'images sur la figure 4 et 5.

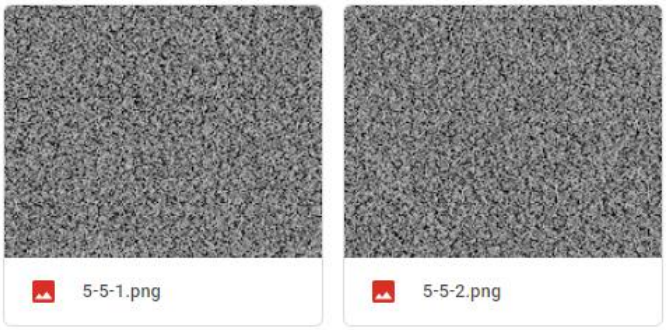


Figure 6 : deux exemples des images d'un texte original- les caractères codés

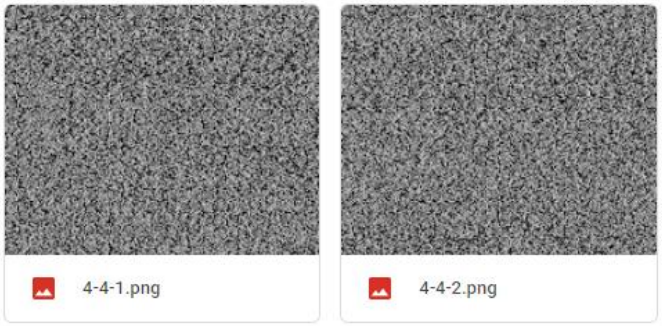


Figure 7 : deux exemples des images d'un texte traduit- les caractères codés

- Les parties du discours

Pour ce type d'images nous avons pris seulement la deuxième couche des images de premier type, qui contient les parties du discours. Ici nous avons gardé le format utilisé pour créer les images en RVB, cela veut dire que nous avons supprimé les espaces. Vous pouvez voir quelques exemples de ce type d'images sur les figures 6 et 7.

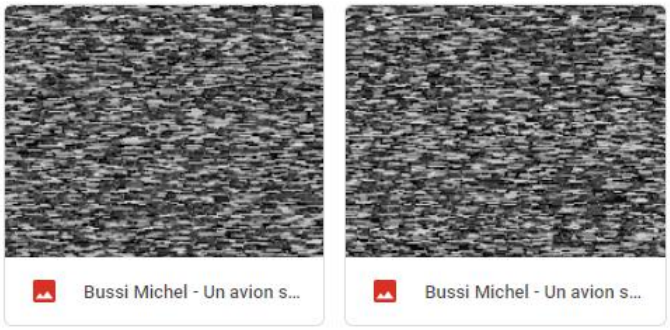


Figure 8 : deux exemples des images d'un texte original- les partie du discours codées

Figure 9 : deux exemples des images d'un texte traduit- les partie du discours codées

- Les fréquences des mots

Finalement pour le dernier type d'images en nivaux de gris nous avons utilisé la

Figure 11 : deux exemples des images d'un texte original- les fréquences des mots codées

Figure 10: deux exemples des images d'un texte original- les fréquences des mots codées

fréquence des mots. Ces fréquences sont étés calculé avec la même méthode que les fréquences des mots du premier type d'images. Pour ce type d'images aussi nous n'avons pas besoin de remettre les espaces car les caractères de chaque mot ont la même valeur dans ce type d'images. Voici deux exemples de chaque catégorie pour ce type d'images :

3.3.4 Apprentissage

Maintenant que nous avons vu les méthodes pour créer les images, nous allons expliquer les méthodes d'apprentissages que nous avons utilisé pour reconnaitre les images des textes traduits et les textes originaux. Comme nous avions mentionné avant, nous avons utilisé un réseau de neurones profonds convolutifs qui s'appelle SqueezeNet et un autre réseau de neurones profond qui s'appelle ResNet.

Ces deux réseaux sont accessibles sur le package du modèle de pytorch. Ils sont disponibles en format pré-entraîné[2] sur les images de ImageNet. Ces deux réseaux neurones ont une couche linéaire comme leur dernière couche et comme ils sont pré-entrainés c'est recommandé de fixer les poids des couches cachées et changer

[2] *Pre -trained*

seulement les poids de la couche linaire, c'est ce que nous avons fait ici. Dans la partie prochaine nous allons voir les résultats.

Nous savons qu'un texte est une structure complexe de mots et leur emplacement alors une image peut être une bonne représentation pour montrer ces complexités.

4. Etude expérimentale

4.1 Méthode « naïves »

Les résultats obtenus par cette méthode sont acceptables, sans être non plus très satisfaisant : la précision varie obtenue varie en moyenne de 60 à 90%, et cette précision semble diminuer significativement quand on le teste sur des textes fortement différents des textes d'entrainement.

4.2 Méthode vectorielle

Nous avons effectué différents essais comme vous pouvez le voir sur les figures disponibles en annexe.

Le modèle optimal semble donc être : découper les textes en morceau de K=50000 caractères, une vectorisation (CountVectorizer [3]) avec les paramètres par défaut (découpe mot par mot, sans stopword,), puis un MultinomialNB[4] avec là aussi aucune modification des hyperparamètres.

Néanmoins nous avons pensé que les vecteurs construits à l'aide de découpages alternatifs pourraient ajouter de l'information supplémentaire, et, ainsi aboutir sur un modèle combinant différentes découpes en regroupant les meilleures prédictions, les synthétisant au mieux et ainsi terminer avec un ultime modèle certes plus couteux et plus lent, mais plus précis.

[3] *https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html*

[4] *https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html*

Observation : Prédiction	Natif	Traduit
Natif	702	5
Traduit	9	715

Figure 12 Exemple de résultat avec la méthode vectorielle (précision d'environ 99%)

Ci-dessus nous pouvons voir une matrice de confusion du modèle que nous avons estimé un des meilleurs sur notre base de textes. Ce modèle est le plus satisfaisant que nous avons pu établir avec des précisions oscillantes habituellement entre 98% et 99.5%. Les temps d'exécution sont plutôt rapides, mais si nous voulons augmenter la précision, il est possible de combiner différents modèles ayant appris sur des découpages différents.

4.3 Images

Maintenant que nous avons parlé de tous qui est nécessaires nous pouvons montrer les résultats des méthodes basées sur les images.

Notre base de données a environ 10.000 images de chaque catégorie pour chaque méthode. Pour la partie d'apprentissage nous avons pris 20 %des images pour validation et 80% pour l'apprentissage.

En utilisant les réseaux dont nous avons parlés dans la partie précédente, pour les images en RVB, la meilleure précision obtenue sur l'ensemble d'apprentissage est 70% et 66% sur l'ensemble de validation qui est obtenue par la premier type d'image et ResNet. Pour les images en niveaux de gris les meilleurs résultats par ordre sont 57.98%, 55.75% et 70% sur l'ensembles d'apprentissage et 57.7%, 55.75% et 67.7% sur l'ensemble de validation. Vous pouvez trouver les courbes d'apprentissage en annexe.

5. Conclusion

Pour résumer, nous voulions mettre en œuvre une méthode de classification de textes, qui permettrait de distinguer des textes traduits de textes natifs. En commençant nous n'étions même pas sûr de la faisabilité d'une telle chose, car nous étions débutant en apprentissage machine et n'avions jamais résolu de problèmes en traitement automatique du langage.

Nous avons finalement proposé deux méthodes principales :

- Une méthode vectorielle
- Une méthode par image

La première méthode a été un franc succès, nous avons atteint des précisions tout à fait satisfaisantes avoisinant les 98-99.5%. Elle est peu couteuse, simple à implémenter et à comprendre et elle offre un potentiel d'explicabilité assez grand bien qu'inexploité ici.

La seconde, quant à elle, fut bien plus compliqué à mettre à place, de par la complexité et la durée des traitements nécessaires. Nous sommes finalement restés sur un échec avec des résultats qui pouvait monter jusqu'à 70%, mais qui ne semblait pas bien généraliser et était très instable d'un apprentissage à l'autre.

Néanmoins, comme la méthode vectorielle nous a prouvé la faisabilité de la tâche, il existe peut-être une manière de modéliser le texte sous forme d'images qui puisse efficacement être utilisé par un réseau de neurone profond pour aboutir à des résultats excellents, nous avons été toutefois incapable de trouver une telle modélisation.

Les axes d'améliorations sont donc premièrement de trouver une bonne modélisation de textes sous forme d'images. Deuxièmement comme nous venons de le mentionner, extraire certains paramètres du modèle vectoriel pourrait être intéressant pour comprendre comment le modèle fonctionne et ainsi accroitre la compréhension que nous avons. Enfin il serait également intéressant d'étendre ce projet à une reconnaissance de textes traduits depuis une myriade d'autres langues et de ne pas se limiter à l'anglais, de même nous pourrions tenter d'appliquer nos méthodes dans des langues différentes et voir si cela fonctionnerait toujours.

6. Bibliographie

Cyril Goutte, P. I. (2009, Janvier). *Automatic Detection of Translated Text and its Impact on Machine Translation.*

Qian Li, H. P. (2022, Avril 8). *A Survey on Text Classification: From Traditional to Deep Learning.* Récupéré sur https://dl.acm.org/doi/10.1145/3495162#d1e4517

Russel, S. J., & Norvig, P. (2010). *Artificial Intelligence - A Modern Approach, Third International Edition.* Pearson Education.

Schmid, H. (1994). Probabilistic Part Of Speech Tagging Using Decision Trees. *Proceedings of International Conference on New Methods in Language Processing*, 9.

7. Annexes

' ': 0	'\n': 1	'!': 3	'"': 4	'#': 5	'$': 6	'%': 7	'&': 8
"'": 9	'(': 24	')': 10	'*': 11	'+': 12	',': 13	'-': 14	'.': 12
'/': 16	':': 17	';': 18	'<': 19	'=': 20	'>': 21	'?': 22	'÷': 23
'@': 24	'[': 25	'\\': 26	']': 27	'^': 28	'_': 29	'`': 30	'{': 31
'|': 32	'}': 33	'~': 34	'€': 35	'£': 36	'¦': 37	'¨': 38	'©': 39
'«': 40	'®': 41	'¯': 42	'°': 43	'±': 44	'´': 45	'·': 46	'¸': 47
'»': 48	'-': 49	'…': 50	'—': 51	'0': 52	'1': 53	'2': 54	'3': 55
'4': 56	'5': 57	'6': 58	'7': 59	'8': 60	'9': 61	'A': 70	'a': 72
'à': 74	'á': 76	'â': 78	'ä': 80	'æ': 82	'B': 84	'b': 86	'C': 88
'c': 90	'ç': 92	'D': 94	'd': 96	'E': 98	'e': 100	'è': 102	'é': 104
'ê': 106	'ë': 108	'F': 110	'f': 112	'G': 114	'g': 116	'H': 118	'h': 120
'I': 122	'i': 124	'î': 126	'ï': 128	'J': 130	'j': 132	'K': 134	'k': 136
'L': 138	'l': 140	'M': 142	'm': 144	'N': 146	'n': 148	'O': 150	'ò': 152
'o': 154	'ó': 156	'ô': 158	'ö': 160	'œ': 162	'P': 164	'p': 166	'Q': 172
'q': 174	'R': 176	'r': 178	'S': 180	's': 182	'T': 184	't': 186	'U': 188
'ü': 190	'ù': 192	'ú': 194	'û': 196	'u': 198	'V': 200	'v': 202	'W': 204
'w': 206	'X': 208	'x': 210	'Y': 212	'y': 214	'Z': 216	'z': 218	

Tableau 1: les codes utilisés pour coder les caractères

Valeur attribuée	Nom de POS (TreeTagger)	Nom grammatical
0	'SPACE'	space
7	'ABR'	abbreviation
14	'ADJ'	adjective
21	'ADV'	adverb
28	'DET:ART'	article
35	'DET:POS'	possessive pronoun (ma, ta, ...)
72	'INT'	interjection
79	'KON'	conjunction
56	'NAM'	proper name
63	'NOM'	noun
70	'NUM'	numeral
77	'PRO'	pronoun
84	'PRO:DEM'	demonstrative pronoun
91	'PRO:IND'	indefinite pronoun
98	'PRO:PER'	personal pronoun
105	'PRO:POS'	possessive pronoun (mien, tien, ...)
112	'PRO:REL'	relative pronoun
119	'PRP'	preposition
126	'PRP:det'	preposition plus article (au,du,aux,des)
133	'PUN'	punctuation
170	'PUN:cit'	punctuation citation
147	'SENT'	sentence tag
154	'SYM'	symbol
161	'VER:cond'	verb conditional
168	'VER:futu'	verb futur
175	'VER:impe'	verb imperative
182	'VER:impf'	verb imperfect
189	'VER:infi'	verb infinitive
196	'VER:pper'	verb past participle
203	'VER:ppre'	verb present participle
210	'VER:pres'	verb present
217	'VER:simp'	verb simple past
224	'VER:subi'	verb subjunctive imperfect
231	'VER:subp'	verb subjunctive present[5]

Tableau 2 : les parties du discours de TreeTagger et leur valeur distribuée

[5] *Source: https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/french-tagset.html*

Figure 15: la courbe d'apprentissage sur les données d'apprentissage (à gauche) et sur les données de validations (à droite) sur les image avec les trois couches- la première méthode

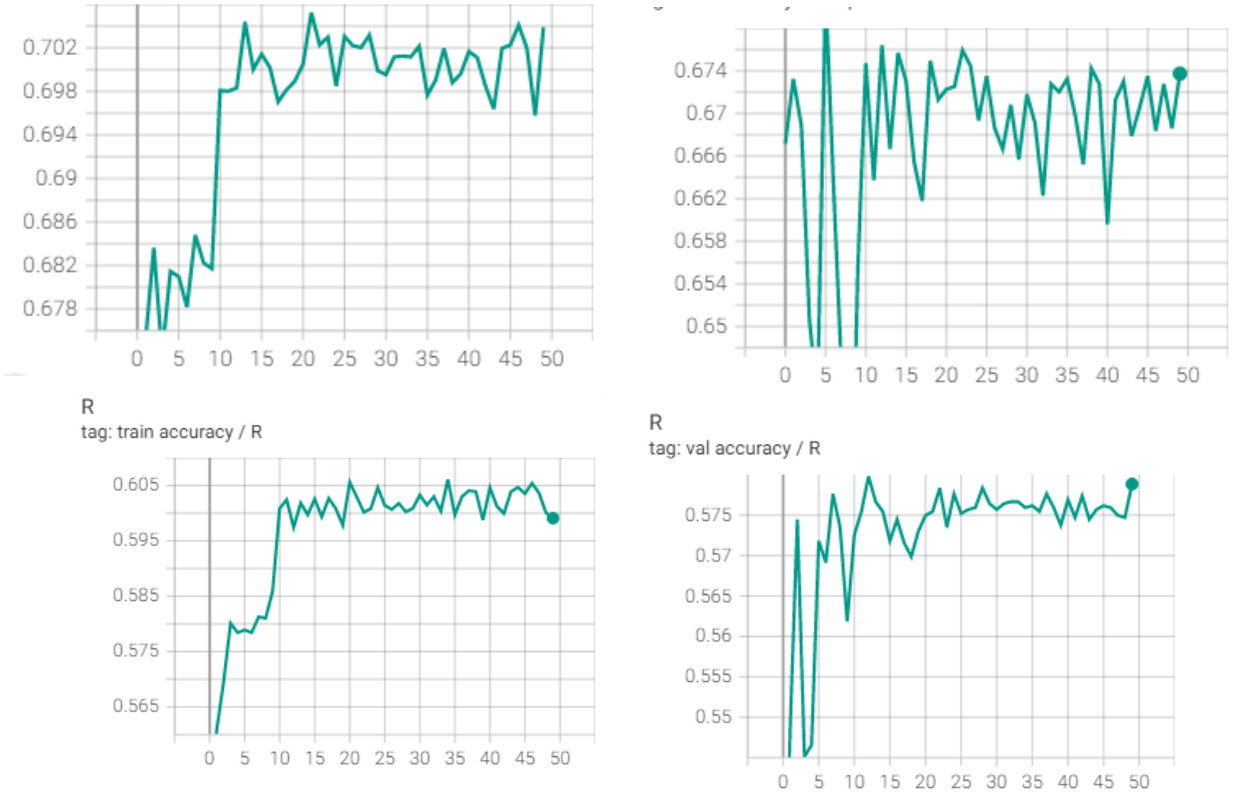


Figure 14 : la courbe d'apprentissage sur les données d'apprentissage (à gauche) et sur les données de validations (à droite) sur les images avec les caractères codés

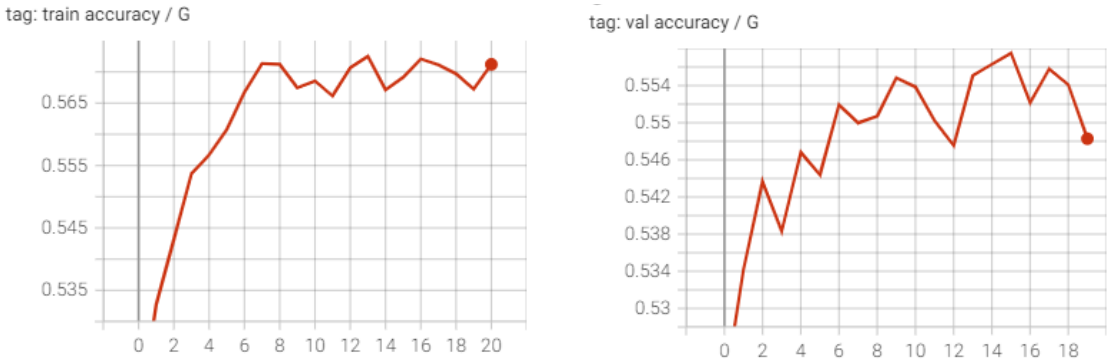


Figure 13: la courbe d'apprentissage sur les données d'apprentissage (à gauche) et sur les données de validations (à droite) sur les images avec les parties de discours

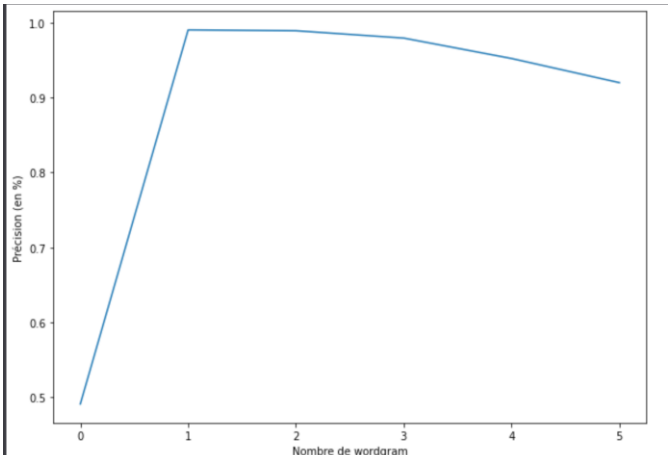


Figure 16 Relation entre le nombre de mots utilisés dans le découpage et la précision du modèle vectoriel

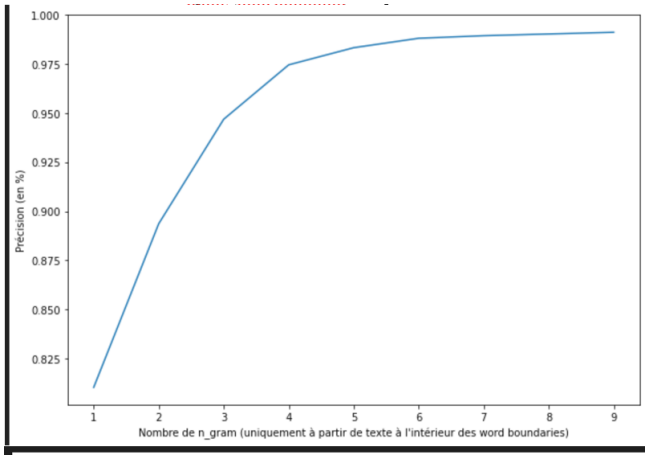
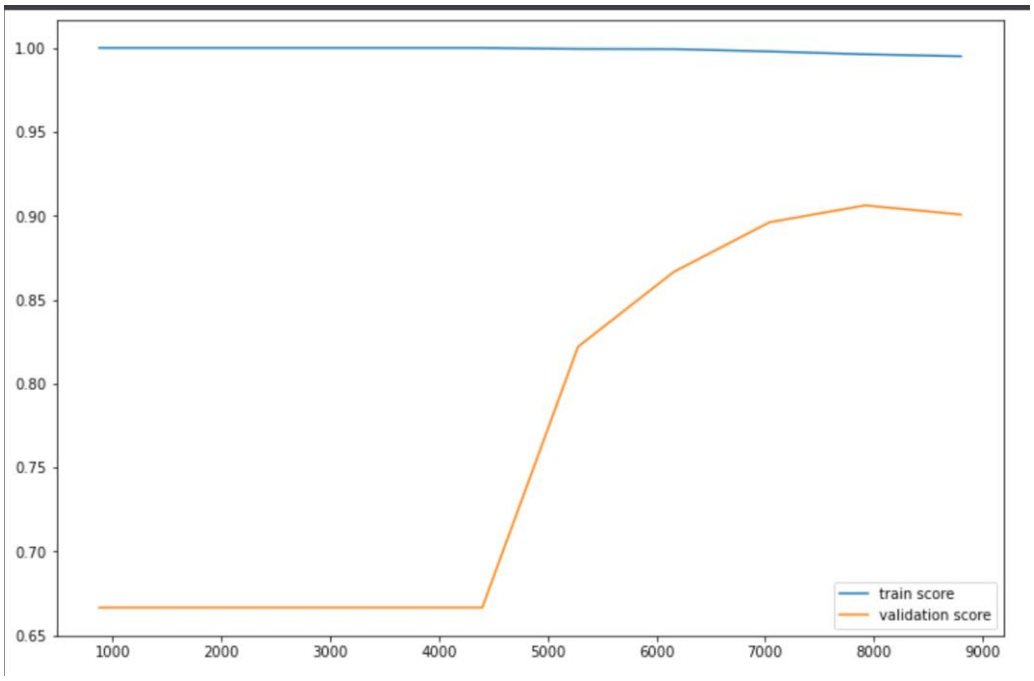


Figure 17 Relation entre le nombre de ngrams utilisés dans le découpage et la précision du modèle vectoriel




```
0.9898181818181818
[[1379   15]
 [  13 1343]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1394
           1       0.99      0.99      0.99      1356

    accuracy                           0.99      2750
   macro avg       0.99      0.99      0.99      2750
weighted avg       0.99      0.99      0.99      2750
```

Figure 18 Exemple de résultats avec le modèle vectoriel en cross validation avec des morceaux de textes de 50.000 caractères