



Formation Ingénieur Machine Learning

Projet 7

Développez une preuve de
concept

Adib Ouayjan

Mentor : M. Lombard

I. Introduction

L'analyse de données textuelles est le processus qui permet d'obtenir des informations précieuses à partir de textes. Les méthodes de *Machine Learning* permettent d'extraire de différents types d'information textuelle en observant des *patterns*. Les résultats de l'analyse de ces *patterns* permettent d'estimer, au titre d'exemple, le pourcentage de commentaires positifs des clients sur un site, la classification des mails reçus en spam ou non et la proposition automatique des tags à partir des textes publiés sur un site etc. (Gavrilova, 2020).

Le projet 7 de la formation "Ingénieur Machine Learning" chez OpenClassrooms vise à effectuer une recherche bibliographique et à mettre en pratique un nouvel algorithme de façon autonome. L'objectif est donc de réaliser une veille thématique pour trouver des sources pertinentes et récentes sur les avancées dans le domaine étudié. Ainsi, une étape de recherche bibliographique a été appliquée pour choisir et implémenter une nouvelle approche. Ensuite, cette dernière a été comparée à une méthode *baseline*.

La *baseline* de cette étude est un modèle que j'ai implémenté précédemment dans le projet 5 de la formation. Ce modèle a pris la forme d'un algorithme de *machine learning* de classification (*SGDClassifier*) et il m'a permis d'analyser des données textuelles afin de proposer automatiquement des tags. Ainsi, l'objectif est d'implémenter une nouvelle approche de proposition de tags. La méthode choisie pour atteindre cet objectif est basée sur l'article de Kumar et *al.* (2020) et celui de Areej et *al.* (2021) sur la proposition automatique des tags. L'algorithme a été implémenté en se basant sur l'étude de Wang et *al.* (2020).

II. Matériels et Méthodes

Pour pouvoir développer un modèle permettant de proposer des tags j'ai utilisé le langage de programmation Python. Tout d'abord, j'ai commencé par récupérer le corpus de texte à partir de "Stack Overflow", un site célèbre de questions-réponses liées au développement informatique, via "StackExchange explorer". Ensuite, un nettoyage et un prétraitement des données (*preprocessing*) ont été appliqués. Une fois les données ont été traitées, j'ai implémenté la *baseline* et la nouvelle approche pour proposer automatiquement des *Tags*. Enfin, la performance de la *baseline* a été comparée avec celle de la nouvelle approche.

II.1. Importation et pré-traitement des données

Les données ont été importées depuis "StackExchange explorer" (**Fig. 1**). Il s'agit de 50 000

questions, ainsi que les descriptions, les tags et les nombres de vues correspondants pour chaque question. La même méthode a été appliquée pour récupérer un nouveau jeu de données. Ce dernier est représenté sous forme de 50 000 questions avec les tags correspondants qui ont été sélectionnés sur des dates différentes de celles du jeu de données principales. Le nouveau jeu de données permet de tester le nouvel algorithme et la nouvelle approche choisie.

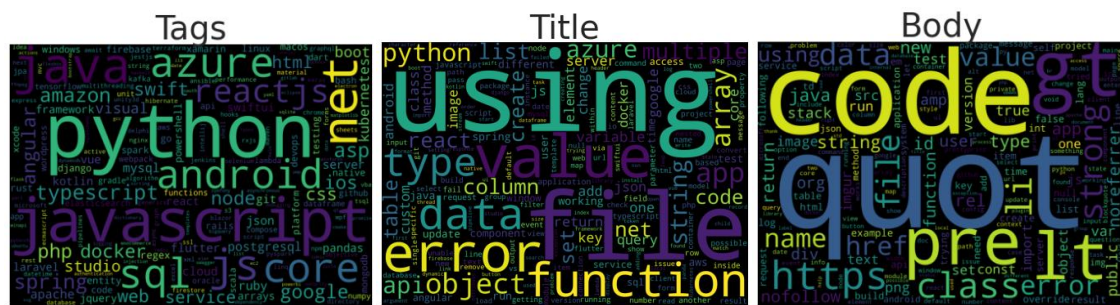


Figure 1. Le code SQL qui a été appliqué sur StackExchange pour récupérer les données d'entraînements de questions (Titles) posées sur Stack Overflow, de la description (Body) ainsi que les Tags correspondants.

Plusieurs méthodes de prétraitement ont été appliquées pour nettoyer et traiter les données afin de les adapter comme *input* des modèles. Ainsi, sur les 50 000 questions téléchargées, seulement celles avec un nombre de vue supérieure à 30 ont été sélectionnées. Le pré-traitement des données a été appliqué sur les questions, les descriptions et les tags. En outre, le corpus texte sur lequel les modèles ont été entraînés sont les questions ; les descriptions ; et les questions et les descriptions prises ensemble. La variable cible est les tags.

Une fois le corpus de texte a été sélectionné, j'ai procédé à la partie pré-traitement du texte. Le pré-traitement du texte signifie simplement le rendre plus propice à l'analyse et à des résultats fiables (Gavrilova, 2020). En premier lieu, un prétraitement simple a été effectué sur le contenu des questions. Pour ce faire, toutes les ponctuations et les mots inutiles ont été supprimés. En effet, les termes comme application, mémoire ou encore Python, Windows, Google etc. sont les plus informatifs pour notre problématique de proposition de tags (Manish, 2018).

Une tokenisation et une lemmatisation ont été appliquées ultérieurement pour transformer chaque question en une liste de mots et pour supprimer complètement les caractères inutiles (Andrade, 2021 ; Kapadia, 2019 ; Jabeen, 2018). La librairie qui a été utilisée pour accomplir toutes ces tâches, est **nlTK**. Les résultats des nuages de mots après pré-traitement des questions, des tags et des descriptions sont représentés dans la **Fig.2**.



d'avoir une représentation similaire. En fait, c'est une classe de techniques où les mots individuels sont représentés comme des vecteurs à valeurs réelles dans un espace vectoriel prédéfini. Chaque mot est mis en correspondance avec un vecteur et les valeurs vectorielles sont apprises d'une manière qui ressemble à celle d'un réseau de neurones (Brownlee, 2017). La position d'un mot dans l'espace vectoriel est apprise à partir du texte et est basée sur les mots qui entourent ce dernier. De cette façon, la "signification" d'un mot peut être reflétée dans son plongement, un modèle est alors capable d'utiliser cette information pour apprendre la relation entre les mots (**Fig.3**). L'avantage de cette méthode est qu'un modèle entraîné sur le mot "House" sera capable de réagir au mot "Home" même s'il n'a jamais vu ce mot durant l'entraînement (Ducan, 2019).

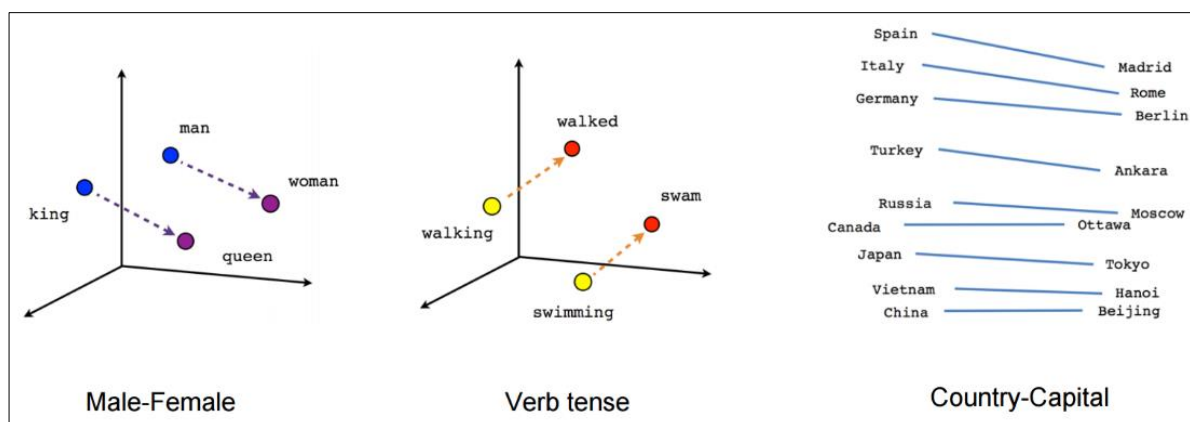


Figure 3. Représentation de la régularités linguistiques dans un espace vectorielle par le *Word embedding* (Chaouche Y., cours OpenClassrooms).

Les plongements de mots peuvent être entraînés en utilisant le corpus d'entrée lui-même ou peuvent être générés en utilisant des incorporations de mots pré-entraînées telles que *Glove*, *FastText* et *Word2Vec* (Shivam, 2018 ; Brownlee, 2017). Dans cette étude, j'ai appliqué un *Embedding Layer* qui a été utilisée dans un modèle de *deep learning* où le plongement est appris en même temps que le modèle lui-même. Elle nécessite que les données d'entrée soient codées en entier, afin que chaque mot soit représenté par un entier unique. Cette étape a été effectuée à l'aide de *Tokenizer* de *Keras*. Pour l'*embedding layer*, la bibliothèque *Keras* propose une couche d'*embedding* qui peut être utilisée pour les réseaux de neurones des données textuelles. Elle est définie comme la première couche cachée (*Hidden Layer*) d'un réseau de neurones (Brownlee, 2017). Elle doit spécifier les 3 arguments suivants : *input_dim* (la taille du vocabulaire dans les données textuelles) ; *output_dim* (la taille de l'espace vectoriel dans lequel les mots seront plongés) ; et *input_length* (la longueur des séquences d'entrée, comme nous le définissons pour toute couche d'entrée d'un modèle *Keras*). Les valeurs de *input_dim* et de

input_length ont été calculées en fonction des données d'entrée et celle de output_dim a été réglée dans le modèle de *deep learning*. Enfin, les principales différences entre la vectorisation par TF-IDF et par *Word embedding* sont présentées dans le **tableau 1**.

Tableau 1. Différence entre *Word Embedding* et TF-IDF traduit à partir du tutoriel de Ducan (2019).

<i>Word Embedding</i>	TF-IDF
Vecteur multidimensionnel qui tente de saisir la relation entre un mot et d'autres mots	Matrice clairsemée où chaque mot ne correspond qu'à une seule valeur et ne capture aucune signification
Souvent entraîné sur un grand corpus externe	Entraîné sans données externes
Doit être appliqué à chaque mot individuellement	Peut être appliqué à chaque document d'entraînement en une seule fois
Plus gourmand en mémoire	Moins gourmand en mémoire
Idéal pour les problèmes impliquant un seul mot	Idéal pour les problèmes comportant de nombreux mots et des documents plus volumineux

II.3. Modélisation

La modélisation des données Stack Overflow a été faite par des approches d'apprentissages supervisés. L'apprentissage supervisé est une approche de *machine learning* qui se définit par l'utilisation d'ensemble de données étiquetées. Ces ensembles de données sont conçus pour entraîner les algorithmes, classer les données et prédire les résultats avec précision. En utilisant des entrées et des sorties étiquetées, le modèle peut mesurer sa précision et apprendre au fil du temps. Dans ce projet, les étiquettes des données sont les tags proposés par Stack Overflow.

Les questions (X), les descriptions (X) et les tags (Y) ont été divisés en 70% de données d'entraînements et 30% de données tests avec la fonction *IterativeStratification*. En effet, dans les données Stack Overflow, les tags sont multi-étiquetés (*multi label*). Ainsi, pour éviter toute erreur liée au texte multi-étiqueté, j'ai choisi de diviser (*split*) les données avec la fonction *IterativeStratification* de *skmultilearn.model_selection*.

Ensuite un modèle de classification linéaire a été implémenté. Il s'agit de l'algorithme *SGDClassifier* de la librairie *sklearn*. Ce modèle a servi comme une *baseline* dans cette étude. La performance de *SGDClassifier* a été comparée à celle d'un modèle de *deep learning*, précisément un réseau de neurones convolutionels (CNN). Le CNN a été choisi suite aux recommandations de l'étude de Wang et al. (2020) et la méthode a été inspirée des deux articles de Kumar et al. (2020) et Areej et al. (2021). Enfin, le notebook a été développé à partir du tutoriel de Janakiev (s.d.) et celui de Brownlee (2017). Tous les entraînements ont été effectués avec Google Colab sur GPU avec un environnement d'exécution à RAM élevé.

II.3.1. SGDClassifier

Le modèle SGDClassifier utilise les données complètes et résout un problème d'optimisation convexe par rapport aux données. Ainsi, il peut traiter les données par lots et effectue une descente de gradient visant à minimiser la perte attendue par rapport à la distribution des échantillons. De ce fait, SGDClassifier est plus adapté aux données avec un grand nombre d'échantillons, ce qui est le cas dans ce projet.

II.3.2. Convolutional Neural Network (CNN)

Les CNNs sont des réseaux neuronaux artificiels multi-couches capables de détecter des caractéristiques complexes dans des données, comme celles des images et des textes. Les CNNs ont été principalement utilisés dans des tâches de vision par ordinateur tel que la classification d'images, la détection d'objets et la segmentation d'images. Récemment, les CNNs ont été appliqués aux problèmes de texte. Un réseau de neurones convolutionnels est composé des couches suivantes (**Fig.4**) : une couche de convolution pour obtenir des caractéristiques (*features*) à partir des données ; une couche de *pooling* pour réduire la taille de la carte des caractéristiques ; une couche *flatten* pour convertir la carte des features et les regrouper en une seule colonne qui sera transmise à la couche *fully connected* ; et une couche *fully-connected* qui reçoit en entrée la carte des features aplaties (*flatten*) et dont la dernière couche renvoie le vecteur des probabilités finales associées à chaque classe en utilisant une fonction d'activation. Dans le cas d'une classification multi-labels, il s'agit d'une fonction d'activation sigmoïde pour une classification binaire (Mwiti, s.d. ; Ashley, 2020).

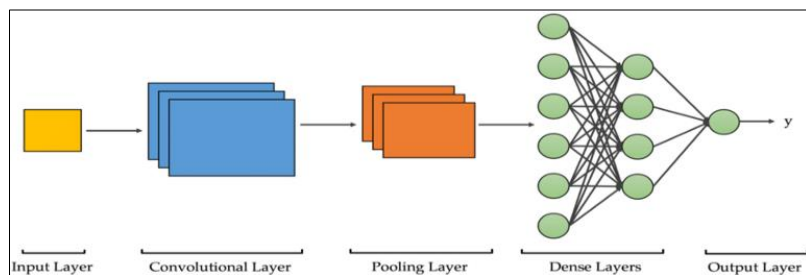
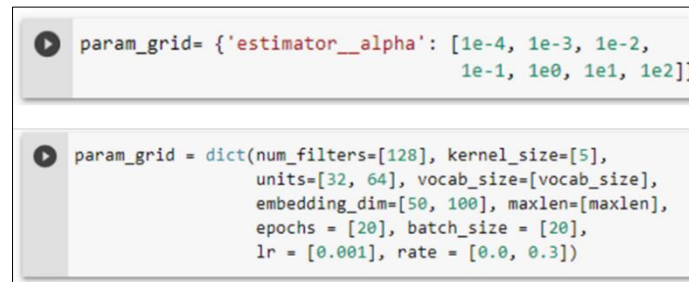


Figure 4. Architecture d'un réseau de neurones convolutionnels, CNN (Ashley, 2020)

II.3.3. Evaluation des modèles

Un *GridSearchCV* a été appliqué et un *accuracy score* a été estimé sur une série de tests de sensibilité dans le but de déterminer les hyperparamètres optimaux correspondants à chaque modèle. L'hyperparamètre alpha a été optimisé pour SGDClassifier avec une cross validation sur 10 folds. Cependant, pour le modèle CNN (*word embedding*) plusieurs hyperparamètres ont

été optimisés avec une *cross validation* sur 2 folds (**Fig.5**). Enfin, l'évaluation des modèles a été faite à l'aide de l'*accuracy* qui calcule la précision de la prédiction sur les données test.



```
param_grid= {'estimator__alpha': [1e-4, 1e-3, 1e-2,
                                   1e-1, 1e0, 1e1, 1e2]}
```

```
param_grid = dict(num_filters=[128], kernel_size=[5],
                  units=[32, 64], vocab_size=[vocab_size],
                  embedding_dim=[50, 100], maxlen=[maxlen],
                  epochs = [20], batch_size = [20],
                  lr = [0.001], rate = [0.0, 0.3])
```

Figure 5. Optimisation des hyperparamètres pour le modèle SGDClassifier (en haut) et CNN (en bas).

II.4. Implémentation de la nouvelle approche

Dans l'étude de Kumar et *al.* (2020) la problématique principale discutée est celle de la rareté des données dans la représentation des tweets dans Twitter. Ainsi, les auteurs ont proposé une nouvelle méthode de recommandation de hashtags qui résout le problème de la rareté des données. Cela a été fait en exploitant des informations plus pertinentes des tweets provenant de sources de connaissances externes. En plus des caractéristiques lexicales, la méthode proposée incorpore des caractéristiques sémantiques basées sur les *word-embeddings*. En effet, contrairement aux documents volumineux, les tweets sont courts et ils ne contiennent pas suffisamment de termes, ce qui représente un grand défi pour la recommandation de hashtags. Comme il y a très peu de contexte dans les tweets, il est essentiel d'obtenir plus d'informations contextuelles à partir de sources externes pour pouvoir recommander de meilleurs hashtags. Pour cela, les auteurs ont extrait les informations contextuelles à partir de sources externes. Ces informations fournissent une représentation alternative et plus riche qu'un tweet.

Par analogie avec ce projet, nous trouvons la même problématique de rareté de contextes dans les questions de Stack Overflow sur laquelle la *baseline* a été entraîné. Cependant, contrairement à l'étude de Kumar et *al.* (2020) les sources externes dans ce projet ont été extraites à partir des descriptions (Body) du Stack Overflow et non pas à partir d'autres sites de *Blog*. Ce choix a été fait en se basant sur l'hypothèse qu'il y a plus d'informations contextuelles dans les descriptions que dans les questions de Stack Overflow. Pour cela, les modèles, SGDClassifier et CNN, ont été entraînés sur les questions, ensuite sur les descriptions et sur les questions et les descriptions prises ensemble. Enfin, les modèles entraînés sur toutes les données (questions + descriptions) ont été testés sur un nouveau jeu de données de questions. La comparaison de la performance de tous ces modèles a été faite sur la base de l'*accuracy*.

III. Résultats

III.3.1. SGDClassifier (TF-IDF)

L'*accuracy score* a été calculé sur le jeu de données test pour tous les modèles SGDClassifier où la vectorisation des données a été faite avec TF-IDF (**Fig.6**). Les résultats montrent que le modèle qui présente la meilleure performance d'*accuracy score* est celui qui a été entraîné sur les questions (*Titles*), avec un score égale à 30.7%. Les classifieurs qui ont été entraînés sur les descriptions (*Body*) et sur les questions et descriptions (*Titles & Body*) présentent respectivement des scores de 26.5% et 29.5%. Le score minimal de 10.6% est celui du classifieur qui a été entraîné sur le jeu de données de questions et descriptions (*Titles & Body*) et testé sur un nouveau jeu de données de questions.

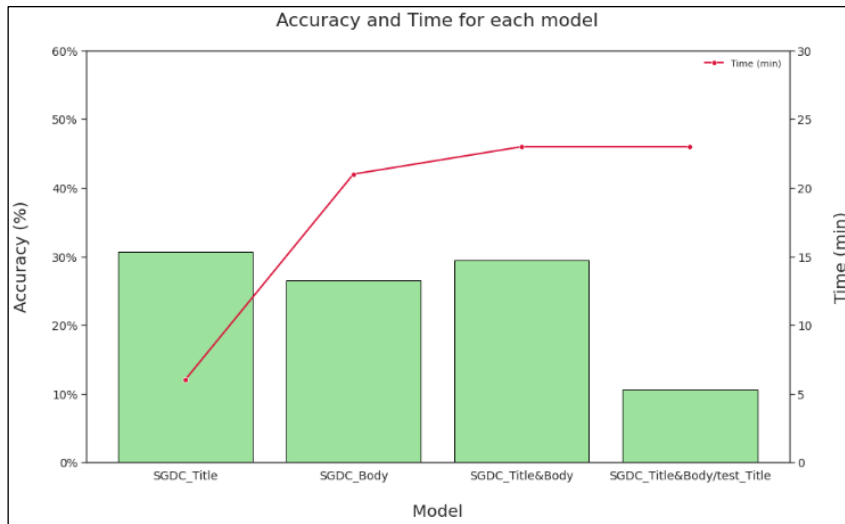


Figure 6. *Accuracy score* (%) et temps (min) des SGDClassifier entraînés sur les questions (SGDC_Title), les descriptions (SGDC_Body), les questions & descriptions (SGDC_Title&Body), les questions & descriptions et testé sur un nouveau jeu de données de questions (SGDC_Title&Body/test_Title).

III.3.2. Convolutional Neural Network, CNN (word embedding)

L'*accuracy score* a été calculé sur le jeu de données test pour tous les réseaux de neurones convolutionnels où la vectorisation des données a été faite avec un plongement des mots (*word embedding*) qui a été appris en même temps que le modèle lui-même (**Fig.7**). Les résultats montrent que le CNN qui présente la meilleure performance d'*accuracy score* est celui qui a été entraîné sur les questions et descriptions (*Titles & Body*) et testé sur un nouveau jeu de données de questions, avec un score égale à 50.9%. Les CNNs qui ont été entraînés sur les descriptions (*Body*) et sur les questions et descriptions (*Titles & Body*) présentent respectivement des scores de 39.4% et 41.5%. Le score minimal de 30.9% est celui du CNN qui a été entraîné sur les questions (*Titles*) seulement.

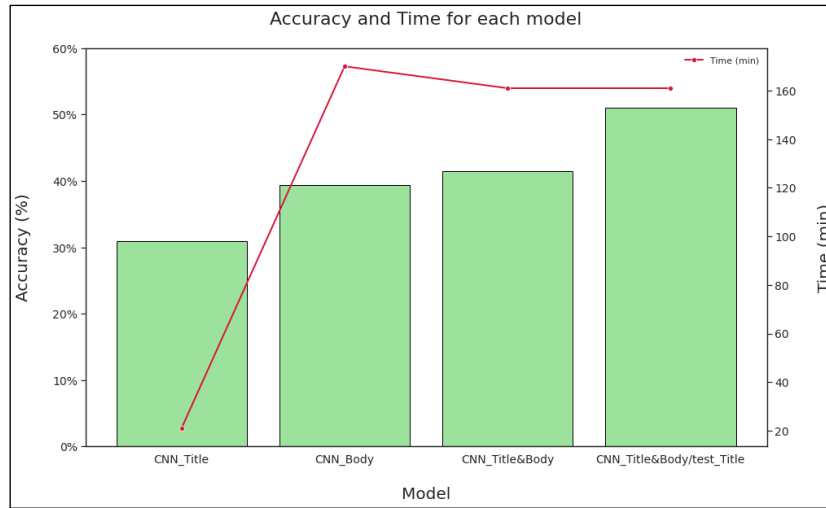


Figure 7. Accuracy score (%) et temps (min) des CNNs entraînés sur les questions (CNN_Title), les descriptions (CNN_Body), les questions & descriptions (CNN_Title&Body), les questions & descriptions et testé sur un nouveau jeu de données de questions (CNN_Title&Body/test_Title).

IV. Discussion

La méthode suivie dans ce projet pour la proposition automatique des tags pour Stack Overflow semble bien fonctionner. Le *Word embedding* inclus dans le réseau de neurones convolutionnels a bien amélioré la performance des modèles. En effet, l'*accuracy* des modèles CNN avec le *word embedding* a augmenté d'une façon significative comparé aux SGDClassifier avec la méthode de vectorisation TF-IDF. Nous notons une augmentation d'*accuracy* de 12.5% pour les modèles qui ont été entraînés sur les descriptions (SGDC_Body vs CNN_Body) et de 12.1% pour ceux entraînés sur les descriptions+questions (SGDC_Title&Body vs CNN_Title&Body). Cette amélioration peut découler du fait que l'approche du *Word embeddings* permet de créer des caractéristiques de texte dans un espace multidimensionnel, plus représentatif de la sémantique des mots. En effet, dans un *embedding*, les mots sont représentés par des vecteurs denses où un vecteur représente la projection du mot dans un espace vectoriel continu, contrairement à TF-IDF où chaque mot ne correspond qu'à une seule valeur dans une matrice clairsemée et ne capture aucune signification (Ducan, 2019 ; Brownlee, 2017). De plus, les méthodes de classification de texte basées sur le *deep learning* peuvent éviter le processus fastidieux d'extraction de caractéristiques (*features*) et présentent une précision de prédiction plus élevée pour un grand ensemble de données non structurées (Wu et al., 2020).

Cependant, le *word embedding* inclus dans le CNN ne semble pas augmenter l'*accuracy* du modèle entraîné sur les questions comparé à la *baseline* (SGDC_Title vs CNN_Title). Cela peut être due à la rareté des données dans les questions. Ce résultat est en accord avec une des conclusions de Wang et al. (2020) sur la performance de CNN qui augmente avec la taille et la

variance des données. Ainsi, avec des données limitées, comme les questions de Stack Overflow, les méthodes traditionnelles d'apprentissage automatique sont assez efficaces en matière d'apprentissage et d'inférence comparés aux modèles profonds comme le CNN. De plus, les résultats présentés dans la **Fig.7** rejoignent la même conclusion sur l'amélioration de la performance des modèles CNNs avec l'augmentation de la taille de données. Nous avons noté une augmentation de l'*accuracy* de 9.5% et de 10.6% pour les modèles CNN_Body et CNN_Title&Body comparés à CNN_Title.

En outre, l'approche de proposition des tags sur les questions inspirée de l'étude de Kumar et *al.* (2020) semble bien marcher. L'*accuracy* du modèle CNN_Title&Body/test_Title entraîné sur les questions+descriptions et testé sur un nouveau jeu de questions est de 50.9%. En effet, si le but de l'étude est de proposer des tags à partir des questions de Stack Overflow (*baseline*) nous rencontrons un problème de la cooccurrence limitée des mots et de la rareté des données dans ces textes courts. Selon Kumar et *al.* (2020) ces problèmes peuvent être surmontés en exploitant des connaissances provenant de sources extrinsèques car elles comblent le fossé sémantique entre les questions et les hashtags associés. Cette amélioration a été observée dans le résultat du modèle CNN_Title&Body/test_Title et non pas sur celui du SGDC_Title&Body/test_Title où l'*accuracy* est minimale de 10.5%. Ce score minimal peut être dû à la nature de la méthode de vectorisation simple et figée de TF-IDF.

En conclusion, ce projet montre que le *word embedding* inclus dans un réseau de neurones convolutionnels améliore les performances des propositions des tags comparés aux méthodes traditionnelles comme la TF-IDF. De plus, l'intégration des connaissances provenant de sources extrinsèques incorpore, en plus des caractéristiques lexicales et topiques, des caractéristiques sémantiques (*semantic features*) basées sur le *word embedding*. L'approche proposée dans ce projet est une bonne piste pour la proposition automatique des tags. Il s'agit d'une approche simple avec un modèle ayant une architecture simple. Cependant, une autre alternative pourrait améliorer la performance de la *baseline*. Elle consiste à utiliser un modèle de *word embedding* pré-entraîné sur un corpus de textes beaucoup plus large. Parmi les méthodes les plus populaires figurent les Word2Vec et BERT, développé par Google, FastText, développé par Facebook, et GloVe développé par Stanford NLP Group (Wang et *al.*, 2020 ; Brownlee, 2017 ; Janakiev, s.d.). Enfin, ces techniques sont performantes mais représentent un outil de modélisation difficile à appréhender. De plus, elles consomment beaucoup plus de temps et de mémoire pendant l'entraînement.

Bibliographie et Webographie

- Andrade (2021). 5 Simple Ways to Tokenize Text in Python. *Towards data science* ([voir ici](#))
- Andrey (2021). What Is Text Vectorization? Everything You Need to Know. *Deepset* ([voir ici](#))
- Areej, Du Q. and Amitava (2021). Hashtag Recommendation Methods for Twitter and Sina Weibo: A Review. *Future Internet* ([voir ici](#))
- Brownlee (2017). How to Use Word Embedding Layers for Deep Learning with Keras (Last Updated on February 2, 2021). *Machine Learning Mastery* ([voir ici](#))
- Brownlee (2017). What Are Word Embeddings for Text? (Last Updated on August 7, 2019). *Machine Learning Mastery* ([voir ici](#))
- Chaouche (s.d.). Analysez vos données textuelles : Effectuez des plongements de mots (*word embeddings*). *OpenClassrooms* ([voir ici](#))
- Ducan (2019). Word Embedding Explained, a comparison and code tutorial. *Medium* ([voir ici](#))
- Gavrilova (2020). Machine Learning & Text Analysis. *Serokell* ([voir ici](#))
- Goyal (2021). Part 5: Step by Step Guide to Master NLP – Word Embedding and Text Vectorization. *Analytics Vidhya* ([voir ici](#))
- Jabeen Hafsa (2018). Stemming and Lemmatization in Python. *Datacamp* ([voir ici](#))
- Janakiev (s.d.). Practical Text Classification With Python and Keras. *Real Python* ([voir ici](#))
- Kapadia Shashank (2019). Building Blocks: Text Pre-Processing. *Towards data science* ([voir ici](#))
- Kumar, Baskaran, Konjengbam & Singh (2020). Hashtag recommendation for short social media texts using word-embeddings and external knowledge. *Knowledge and Information Systems* ([voir ici](#))
- Manish (2018). Entity Extraction Using NLP in Python. *OpenSens Labs* ([voir ici](#))
- Wang, Nulty & Lillis (2020). A Comparative Study on Word Embeddings in Deep Learning for Text Classification. *4th International Conference on Natural Language Processing and Information Retrieval* ([voir ici](#))
- Wu, Liu & Wang (2020). Review of text classification methods on deep learning. *Computers, Materials & Continua* ([voir ici](#))