OPENCLASSROOMS

Formation Ingénieur Machine Learning

Projet 5

Catégorisez automatiquement des questions

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 reçue en spam ou non et la proposition automatique des tags à partir des textes publiés sur un site etc.

Le projet 5 de la formation Ingénieur Machine Learning chez Openclassrooms vise à analyser des données textuelles afin de proposer automatiquement des tags. Les données sont récupérées à partir de 'Stack Overflow', un site célèbre de questions-réponses liées au développement informatique. Pour pouvoir poser une question sur ce site, il faut entrer plusieurs tags de manière à retrouver facilement la question par la suite. Ainsi, le but de ce projet est de développer un modèle de proposition automatique de tags. Ce modèle prendra la forme d'un algorithme de *machine learning* (apprentissage automatique) proposant des tags pertinents à une question. Enfin, le modèle sera validé et déployé en ligne sous forme d'une application.

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 sur "StackExchange explorer". Ensuite, une exploration et un prétraitement des données (*preprocessing*) ont été appliqués. Une fois les données ont été traités, j'ai appliqué plusieurs algorithmes d'apprentissage automatique (*Machine learning*) supervisés et non-supervisés dans le but de trouver un modèle optimal de proposition de tags. Enfin, le modèle sélectionné, a été déployé en ligne sous forme d'un api après avoir été validé sur un jeu de données test.

1. Importation des données

Les données ont été téléchargées depuis "StackExchange explorer" (**Fig. 1**). Il s'agit de 50 000 questions, entre le 10-08-2020 et le 10-08-2021, avec les tags correspondants, ainsi que les nombres de vues pour chaque question. La même méthode a été appliquée pour récupérer le jeu de données test. Ce dernier est représenté sous forme de trente questions choisies aléatoirement

en mois d'octobre de l'année 2021. Le jeu de données test aide à la prise de décision sur la sélection du modèle à déployer sous forme d'api.

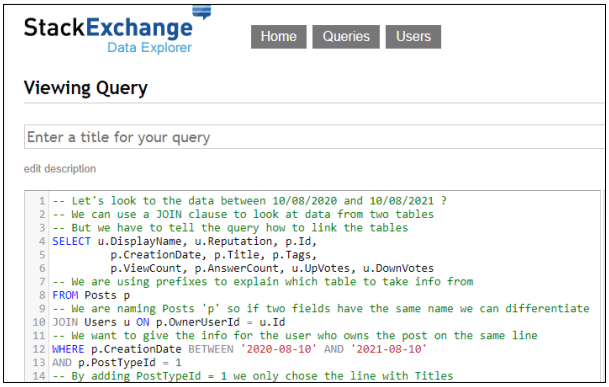

Figure 1. Le code SQL qui a été appliqué sur StackExchange pour récupérer les données d'entrainements de questions posées sur Stack Overflow, ainsi que les tags correspondants.

2. Pré-traitement des données

Dans cette partie, plusieurs méthodes ont été appliqué pour nettoyer et traiter les données afin de les adapter comme *input* des modèles supervisés et non-supervisés. Ainsi, sur les 50 000 questions téléchargées, 5000 seulement ont été sélectionnées avec la seule condition d'avoir un nombre de vue qui est supérieure à 30. En outre, le texte sur lequel les modèles ont été entrainées sont les questions. Ce choix a été fait après une analyse de la distribution des longueurs des questions dans le jeu de données. Le résultat montre que 75% des questions présentent plus de 7 mots (**Fig. 2**). De plus, si on part du principe que le titre résume le texte sous-jacent et qu'il présente souvent les mots-clés les plus importants et les plus pertinents au sujet, cela nous permet de conclure que le titre présente, en lui seule, les informations nécessaires pour alimenter les modèles sans trop les polluer. Pour cela, les colonnes qui ont été sélectionnées pour entrainer les modèles sont les questions et les tags correspondants.

En effet, la réduction de la taille du jeu de données par un facteur de 10 (5 000 questions au lieu de 50 000) et le choix des questions comme étant le seul texte m'ont permis d'optimiser le temps d'implémentation des modèles, ainsi que la mémoire consommée.

Une fois le corpus de texte a été séléctionné, 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. En premier lieu, un prétraitement simple a été éffectué sur le contenu des questions. Pour ce faire, toutes les ponctuations et les mots ont été supprimés sauf les noms (noms propres

inclus). En effet, les termes comme application, mémoire ou encore Python, Windows, Google etc. sont les plus informatifs pour notre problématique de proposition de tags.

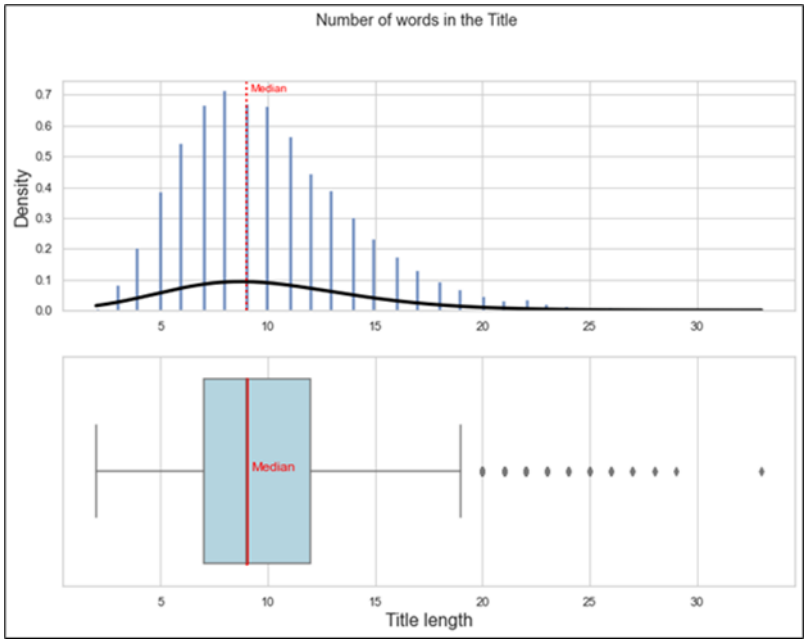

Figure 2. Distributions du nombre de mots dans les questions du jeu de données Stack Overflow.

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. La librairies qui a été utilisée pour accomplir toutes ces tâches est **nltk**. Enfin, le résultat de comparaison de nuage de mots avant et après pré-traitement du texte montre que les termes après le pré-traitement sont plus informatifs pour notre problématique de proposition de tags (**Fig. 3**).

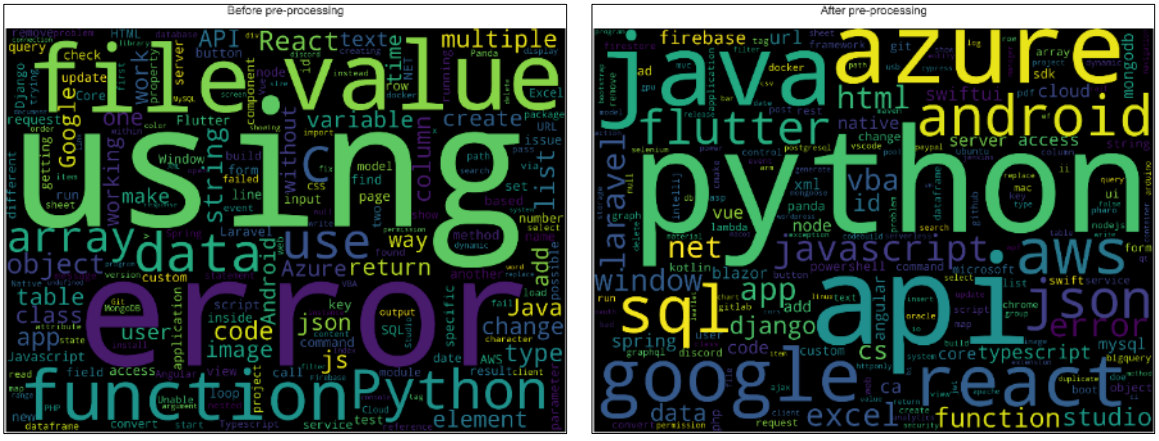

Figure 3. Nuage de mots les plus fréquents dans le corpus texte avant et après pré-traitement. La taille de chaque mot est proportionnelle à sa fréquence dans le corpus.

3. Modélisation

Après avoir préparé les données textuelles j'ai procédé à la partie modélisation. Pour faire cela, le texte doit être vectorizer. La vectorisation de texte consiste à le convertir en une représentation numérique. Cela permet donc d'entrainer le corpus avec des modèles supérvisés et non-supérvisés. Enfin, les différents modèles entrainés seront comparés sur la base de 30 questions choisies aléatoirement en octobre 2021. Le modèle le plus performant, sera donc déployé en ligne à l'aide de Github et de Heroku.

3.1.Modèles non supervisés

'L'apprentissage non supervisé utilise des algorithmes d'apprentissage automatique pour analyser et regrouper des ensembles de données non étiquetées. Ces algorithmes découvrent des modèles cachés dans les données sans nécessiter d'intervention humaine (d'où le terme "non supervisé")'. Dans ce projet, j'ai appliqué deux algorithme d'apprentissage non-supervisés dans le but de proposer automatiquement des tags, *Latent Dirichlet Allocation* (**LDA**) et *Negtaive Matrix Fcatorisation* (**NMF**).

3.1.1. LDA

Le modèle LDA fait partie d'une catégorie de modèles appelés "topic models", qui permet de découvrir des structures thématiques cachées dans des grandes archives de documents. Ceci permet d'obtenir des méthodes efficaces pour l'organisation automatique des documents par sujet. Grâce à ça, nous pouvons proposer des tags à partir de la liste de mots d'un sujet auquel appartienne la question.

Pour le LDA, j'ai utilisé les modèles de bigrammes et trigrammes comme méthode de vectorisation. Un modèle qui se base simplement sur la fréquence d'apparition d'un mot sans tenir compte des mots précédents est appelé unigramme. Si un modèle ne prend en compte que le mot précédent pour prédire le mot actuel, il s'agit d'un modèle bigramme etc. J'ai utilisé le modèle Phrases de la librairie **Gensim** pour construire et implémenter les bigrammes et trigrammes.

Ensuite, la fonction corpora de Gensim a été utilisé pour créer un dictionnaire et un corpus. Gensim crée un corpus en attribuant un identifiant unique pour chaque mot du document et en calculant sa fréquence d'apparition dans ce document. Cette étape est

importante car le corpus et le dictionnaire sont les des deux points d'entrées principales du modèles LDA.

Le LDA a été executé en utilisant *LdaMulticore* de *gensim.models*. Un *coherence score* a été estimé sur une série de tests de sensibilité dans le but de déterminer les hyperparamètres de *LdaMulticore*, Nombre de sujets (K), Hyperparamètre de Dirichlet alpha (densité document/sujet) et beta (Densité mots/sujets).

Le résultat du réglage des hyperparemètres montre que le *coherence score* le plus élevé est d'environ 0,73 pour K=12, alpha = 0,31 et Beta = 0,01.

Enfin, l'une des applications pratiques de LDA, et du *Topic modeling* en général, est de pouvoir déterminer le sujet d'une question (ou document) donnée. Ainsi, pour proposer les tags, j'ai sélectionné les deux sujets qui ont le pourcentage de contribution les plus élevés pour une question. Ensuite, j'ai sélectionné les mots de ces deux sujets qui se trouve dans le texte initial (après pré-traitement). De plus, j'ai ajouté les deux mots les plus fréquents et communs entre ces deux sujets. Tous ces mots ensemble constituent, donc, la liste de de tags pour une question donnée.

3.1.2. Negative Matrix Factorisation

La NMF est une technique non supervisée. Il n'y a donc pas d'étiquetage des sujets sur lesquels le modèle sera entraîné. Le principe de fonctionnement est le suivant : la NMF décompose (ou factorise) des vecteurs de haute dimension en une représentation de plus faible dimension. Ces vecteurs de dimension inférieure sont non négatifs, ce qui signifie également que leurs coefficients sont non négatifs. Ainsi, en utilisant la matrice originale (A), la NMF nous donnera deux matrices (W et H). W représente les sujets trouvés et H les coefficients de ces sujets. Dans notre cas, les vecteurs à haute dimension seront des poids Tf-idf calculés à partir du modèle *TfidfVectorizer* de la librairie sklearn.

La NMF a été executée en utilisant *NMF* de *sklearn.decomposition*. Un *coherence score* a été estimé sur une série de tests de sensibilité dans le but de déterminer le nombre de sujets (num_topics). Le résultat du réglage des hyperparamètres montre que le coherence score le plus élevé est d'environ 0,65 pour num_topics=12.

Enfin, la NMF fait partie des techniques du *Topic modeling*. Pour cela, j'ai suivi la même méthode que celle de LDA pour proposer une liste de tags pour chaque question.

3.2.Modèles supervisés

‘L'apprentissage supervisé est une approche de l'apprentissage automatique qui se définit par l'utilisation d'ensemble de données étiquetées. Ces ensembles de données sont conçus pour entraîner (ou superviser) les algorithmes, à classer les données, ou à 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 de Stack Overflow. Pour cela, un simple pré-traitement leur a été appliqué et qui consiste à la suppression des ponctuations, ainsi qu’à la tokenisation et la lemmatisation.

Les questions (X) et les tags (Y) ont été vectorisés respectivement avec *TfidfVectorizer* et *MultiLabelBinarizer* de la librairie *sklearn*. Ensuite, les données ont été divisés en un 70 % de données d’entrainement et 30% de données tests avec la fonction *IterativeStratification*. En effet, dans les données Stack Overflow, les tags sont multi-étiquetées. Ainsi, pour éviter toute erreur liée au texte multi-étiqueté, j'ai choisi dde diviser (*split*) des données avec la fonction *IterativeStratification* de *skmultilearn.model_selection*.

Enfin, j’ai appliqué quatre algorithmes de classification supervisée dans le but de proposer automatiquement des tags : Logistic Regression, Linear SVC, SGD Classifier et Decision Tree Classifier. Un score F1-micor (micro-moyenne) a été estimé sur une série de tests de sensibilité dans le but de déterminer les hyperparamètres optimaux correspondants à chaque modèle (**Tab. 1**). Le choix de F1-micro a été fait car la micro-moyenne va agréger les contributions de toutes les classes pour calculer la moyenne. Ainsi, dans une configuration de classification multi-classes, la micro-moyenne est préférable s'il y a un déséquilibre entre les classes.

Tableau 1. Réglage des hyperparamétres correspondant aux modèles suivants : Logistic Regression, Linear SVC, SGD Classifier et Decision Tree Classifier.

Model	Hyperpramétres	Tuning values	Optimal value	F1 score
Logistic Regression	C	[0.001, 0.01, 0.1, 1, 10, 100, 1000]	100	0.38
Linear SVC	C	[0.0001, 0.001, 0.01, 0.1, 1, 10]	1	0.43
SGD Classifier	alpha	[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]	0.0001	0.45
Decision Tree Classifier	max_depth	range(1, 10)	8	0.44
	min_samples_split	range(2,10)	8	
	min_samples_leaf	range(1,5)	2	
	criterion	[gini, entropy]	entropy	

4. Déploiement en ligne

Une comparaison de la prefrormance de tous les modèles a été faite sur les données test. Seul le modèle le plus performant sera déployé en ligne à l'aide des plateformes **Github** et **Heroku**. Pour ce faire, j'ai tout d'abord créé une nouvelle application sur Heroku. Ensuite, j'ai initié un dossier de projet sur mon PC local (**Fig. 4**). Une fois l'application web a fonctionné correctement sur mon PC local, j'ai initialisé Git et j'ai poussé (*push*) mon dossier local dans GitHub. Enfin, j'ai connecté Heroku à mon dossier GitHub pour pouvoir déployer automatiquement mon application.

Figure 4. Différents fichiers du dossier de projet sur mon PC local.

III. Résultats

1. Modèles non-supervisés

Les résultats pour le LDA et la NMF montrent que les *coherence scores* sont respectivement 0.73 et 0.65 avec un nombre de sujet optimal qui est égal à 12 pour les deux modèles.

Pour la prédiction des tags, les deux modèles semblent fonctionner correctement pour certaines questions. Cependant, pour une grande majorité des questions la liste des tags est vide pour LDA ou répétitive pour NMF (**Tab. 2**).

2. Modèles supervisés

Le score F1-micro a été calculé sur le jeu donné test (X_test) pour les 4 modèles supervisés (Logistic Regression, Linear SVC, SGD Classifier et Decision Tree Classifier). Les résultats

montrent que le modèle qui présente la meilleure performance en matière de F1-micro est SGD Classifier avec un score égale à 0.47.

La prédiction des tags à partir de SGD Classifier semble bien fonctionner sur une partie des questions (**Tab. 3**).

Tableau 2. Les listes des tags prédites par les modèles non-supervisés, LDA (Tags_LDA) et NMF (Tags_NMF), pour certaines questions de Stack Overflow (Title).

	Title	Tags_LDA	Tags_NMF
0	How can I get puppeteer objects from within a ...	[]	[eks, ec]
1	Why is jq modify and if-then-else not working	[]	[eks, ec]
2	Can GStreamer's rtspsrc element pass authentic...	[webpack, python, vba]	[eks, ec]
3	Remove a Layer from an AWS Lambda Function	[aws, sql, javascript, app]	[aws, lambda, function, eks]
4	spring-webmvc/hibernate-validator not invalida...	[]	[eks, ec]
5	How to use local system fonts with wkhtml2pdf ...	[]	[eks, ec]
6	Unable to access DOM elements in component dur...	[]	[eks, ec]

Tableau 3. Les listes des tags prédites par le modèle supervisé, SGD Classifier (Tags_SGDC) pour certaines questions de Stack Overflow (Title).

	Title	Tags_SDGC
0	How can I get puppeteer objects from within a ...	[]
1	Why is jq modify and if-then-else not working	[]
2	Can GStreamer's rtspsrc element pass authentic...	[]
3	Remove a Layer from an AWS Lambda Function	[amazon, aws, lambda, service, web]
4	spring-webmvc/hibernate-validator not invalida...	[boot, java, spring]
5	How to use local system fonts with wkhtml2pdf ...	[html]
6	Unable to access DOM elements in component dur...	[javascript]

3. Validation des modèles

La comparaison des tags prédits par SGD Classifier, LDA et NMF et ceux proposés par le site Stack Overflow montre que le modèle supervisé semble mieux fonctionner que les non-supervisés (**Tab. 4**). Les résultats montrent que dans certaines cas, les modèles n'arrivent pas à prédire des tags. En effet, les modèles retournent des listes vides pour SGDC et LDA ou une liste de tags redondants pour NMF. Dans les cas où le modèle réussit à prédire des tags, la SGDC propose des tags que nous les trouvons dans la liste de ceux de Stack Overflow.

Cependant, les deux autres modèles, LDA et NMF, semblent moins fonctionné, proposant ainsi, des listes de tags ayant moins de terme en commun avec ceux de Stack Overflow.

Tableau 4. Les listes des tags prédites pour 10 questions (Title) posées sur le site Stack Overflow en octobre 2021. Les tags sont prédits à partir des modèles SGD Classifier (Tags_from_supervised_model) , LDA (Tags_from_unsupervised_LDA) et NMF (Tags_from_unsupervised_NMF). Enfin, les listes des tags proposés par Stack Overflow se trouvent dans la colonne Tags_StackOverflow.

	Title	Tags_StackOverFlow	Tags_from_supervised_model	Tags_from_unsupervised_LDA	Tags_from_unsupervised_NMF
0	npm ERR! Unsupported URL Type "workspace:": wo...	javascript npm dependencies node modules ...	[]	[]	[url, err, eks, ec]
1	How to encrypt a pickled file in Python?	python encryption nlp cryptography pickle	[python]	[]	[python, file, error]
2	How import local Java dependency?	java maven dependency management	[java]	[api, azure, app]	[java, firebase, function]
3	When is an ellipsis needed when dealing with p...	c++ variadic templates parameter pack	[]	[json]	[eks, ec]
4	Is `sizeof(T)` with an incomplete type a valid...	c++ language lawyer sfinae	[]	[]	[eks, ec]
5	Is there a way to prevent click input from bots?	javascript typescript bots pixi js	[py, python]	[mvc]	[input, eks, ec]
6	Springboot test using ConfigurationProperties ...	java spring boot junit spring test	[]	[test]	[eks, ec]
7	Xcode 13 build error - Command PhaseScriptExec...	react native	[]	[exit]	[error, code, build, eks, ec]
8	Ansible playbook not running in GitHub Actions	ansible github actions	[github]	[mvc, span, web]	[github, eks, ec]
9	Powershell scripting - replace text in files	powershell	[powershell]	[powershell, react, wpf, mac]	[file, eks, ec]
10	C# serialization using Json.NET, taking immens...	c# serialization json net deserialization	[json, net]	[dynamodb]	[json, net, system, filebeat]

En conséquence, le modèle SGD Classifier a été sélectionné pour être déployé en ligne. L'application est hébergé par Heroku. Elle permet, donc, de prédire des tags pour une question donnée à partir du modèle SGD Classifier déjà entrainé (**Fig. 5**).

Figure 5. L'application de proposition de Tags pour le site Stack Overflow. Elle est hebergée en ligne sur Heroku https://app-tags-stackoverflow-ao.herokuapp.com/ . À gauche c'est la fenêtre d'entrée (*input*) de l'application avec une question déjà posée. À droite c'est la fenêtre de sortie (*output*) de l'application avec la liste des tags correspondante à la question.

IV. Discussion

La méthode proposée pour le prétraitement et l'analyse du texte semble bien fonctionner. Le nuage des mots, des données prétraitées, présente des termes qui sont plus informatifs que celui

du texte avant pré-traitement (**Fig. 3**). De plus, la sélection des questions comme le seul corpus de texte a permis une réduction de la taille du texte sans perdre beaucoup d'information.

La lemmatisation est un des traitements utiles dans les données textuelles. Elle permet d'obtenir les formes racines des mots infléchis (dérivés). Cependant, cela peut transformer certains mots importants comme *Pandas,* qui est une librairie dans Python, en *panda*. Ainsi, le tag proposé pour une question sur la librairie *Pandas* sera un mot dérivé de ce dernier et non pas le vrai nom de la librairie. L'approche NER (*Named Entity Recognition*) pourrait surmonter ce problème car elle permet d'extraire et de classer les entités nommées dans le texte dans des catégories prédéfinies.

Concernant les performances des modèles en matière de prédiction de tags, les modèles supervisés semblent mieux fonctionner que ceux non-supervisés. Une analyse visuelle de 30 questions choisies aléatoirement en octobre 2021, dont 11 seulement sont présentés dans le **tableau 4**, montre que les trois modèles proposent des tags pertinents aux questions. L'analyse visuelle de ces prédictions avec les listes de tag Stack Overflow montre que SGD Classifier propose des tags communs à ceux du site.

Pour les questions comportant un texte peu informatif, la méthode supervisée (SGD Classifier), prédit des listes vides contrairement aux modèles non-supervisés. En effet, les approches non-supervisées ont tendance à proposer plus souvent des tags. Cependant, les tags proposés par ces derniers sont souvent moins liés aux questions. Nous pouvons noter dans le **tableau 4**, par exemple, que la NMF force la prédiction d'une liste de tags répétitifs ([eks, ec]) au lieu de ne rien prédire.

Par conséquent, le modèle SGD Classifier semble le mieux fonctionner pour mes jeux de données. SGD Classifier utilise les données complètes et résout un problème d'optimisation convexe par rapport aux données. 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, SGD Classifier est plus adapté aux données avec un grand nombre d'échantillons, ce qui est le cas dans ce projet.

Pour toutes ces raisons, seul le SGD Classifier a été déployé en ligne sur Heroku. La facilité d'implémentation et la performance globale de ses prédictions permettent de tourner le modèle en ligne en peu de temps.

Enfin, les résultats montrent, globalement, que la méthode et le modèle proposés répondent à la problématique de base qui est le développement d'un système de suggestion de tag pour Stack Overflow. Cependant, d'autres approches pourront augmenter la performance de ce système de suggestion comme le *Word Embeddings*. Il s'agit d'une représentation apprise pour un texte où les mots ayant la même signification ont une représentation similaire. Techniquement, les méthodes *Word Embeddings* apprennent une représentation vectorielle à valeurs réelles pour un vocabulaire prédéfini et de taille fixe. Le processus d'apprentissage de cette méthode peut être conjoint avec le modèle de réseau neuronal sur une certaine tâche (classification de documents) ou un processus non-supervisé, utilisant les statistiques des documents.

Bibliographie et Webographie

Blei D. M., Ng A. Y., and Jordan M. I.. Latent Dirichlet Allocation. Journal of Machine Learning Research, 3:993–1022, 2003.

Charmaine Chui, Deploy Your Python Machine Learning Models on Heroku in 3 steps. Towards data science consulté en octobre 2021. Deploy Your Python Machine Learning Models on Heroku in 3 steps | by Charmaine Chui | Towards Data Science

Jason Brownlee, What Are Word Embeddings for Text?. Machine Learning Mastery consulté en octobre 2021. https://machinelearningmastery.com/what-are-word-embeddings/

Jason Brownlee, How to Encode Text Data for Machine Learning with scikit-learn. Machine Learning Mastery consulté en octobre 2021. https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

Julianna Delua, Supervised vs. Unsupervised Learning: What's the Difference?. IBM consulté en octobre 2021. https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

Mustafa Murat Arat, Metrics for Multilabel Classification. Github consulté en octobre 2021. https://mmuratarat.github.io/2020-01-25/multilabel_classification_metrics

Rob Salgado, Topic Modeling Articles with NMF. Towards data science consulté en octobre 2021. https://towardsdatascience.com/topic-modeling-articles-with-nmf-8c6b2a227a45

Shirley Chen, Getting Started with Text Vectorization. Towards data science consulté en octobre 2021. https://towardsdatascience.com/getting-started-with-text-vectorization-2f2efbec6685#:~:text=Text%20Vectorization%20is%20the%20process,(L1)%20Normalized%20Term%20Frequency

Susan Li, Named Entity Recognition with NLTK and SpaCy. Towards data science consulté en octobre 2021. Named Entity Recognition with NLTK and SpaCy | by Susan Li | Towards Data Science

Ula La Paris, Named Entity Recognition : la personnalisation de suggestions d'articles tech. Saegus consulté en octobre 2021. https://medium.com/data-by-saegus/ner-medium-articles-saegus-7ffec0f3188c

Yulia Gavrilova, Machine Learning & Text Analysis. Serokell consulté en octobre 2021. https://serokell.io/blog/machine-learning-text-analysis