

**Rapport**

***Projet 5 IML***

**Catégorisez automatiquement des questions**



**Thibault Roussel**

**SOMMAIRE**

1. Introduction
2. Les différents traitements effectués
3. Analyses univariées
4. Les modélisations effectuées :

* Approche supervisée
* Approche non supervisée

1. La modélisation choisie pour la mise en production
2. Reduction dimensionnelle
3. Livrables
4. Synthèse

**1-Introduction**

Créé en 2008, Stack Overflow est un célèbre site web proposant des questions-réponses sur un large choix de thèmes concernant la programmation informatique. Il fait partie du réseau de sites Stack Exchange.

Lorsqu’un utilisateur pose une question sur le site, il doit y associer plusieurs tags afin que la question posée soit aisément retrouvée et éviter ainsi un trop grand nombre de questions en doublons.

Ce reflex est bien pris par les utilisateurs expérimentés, mais moins par les nouveaux.

L’objectif de ce projet est alors de développer un système de suggestions automatique de quelques tags relatifs à la question posée sur le site.

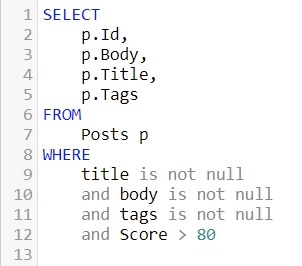
Pour ce faire, nous allons d’abord réaliser un prétraitement sur le texte afin de pouvoir développer des modèles de Machine Learning. Ce prétraitement permettra de passer de chaînes de caractères à des tableaux de mots exploitables de manière optimale par nos algorithmes supervisés ou non.

Pour finir, une API (interface de programmation d’application) proposant automatiquement 3 tags à une question sera développée avec la librairie Flask.

**2-Les différents traitements effectués**

Au début de ce projet, nous avons utilisé l’outil d’export de données - "stackexchange explorer" afin de récupérer les données.

Après plusieurs essais, voici la manière dont nous avons procédé :



Grace au langage de programmation sql, un premier traitement sur le dataset peut ainsi être fait dès cette première étape. Nous avons fait le choix de conserver les questions ayant un titre, un body et au moins un tag ainsi qu’un score supérieur à 80.

Un score élevé permet de s’assurer de la bonne qualité des questions.

Nous obtenons alors 58.903 questions, il n’y a pas de doublons dans Body, mais il y en a 5 dans Titre que nous supprimons.

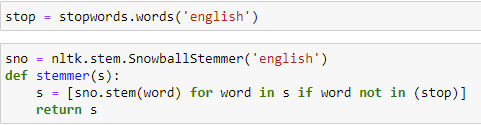
On a ensuite créé des fonctions de prétraitement de Body et Title :

* Une fonction de **tokenisation** :



Cette fonction sépare les chaines de caractères présents dans Title et Body en tableaux de mots afin de pouvoir effectuer des opérations dessus. Elle supprime également les balises html et quelques symboles.

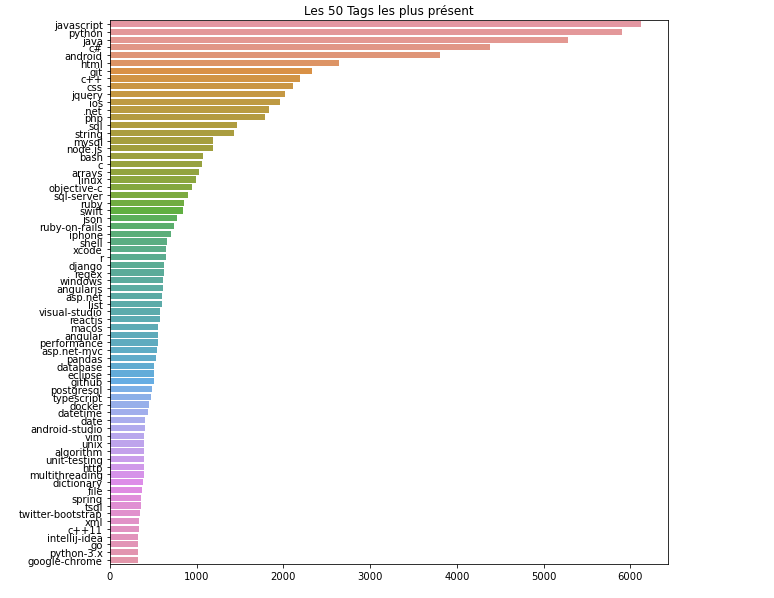
* Une fonction pour le **'stemming'** et le suppression des stopwords :



Le stemming consiste à conserver uniquement la racine des mots étudiés. Cette fonction, en plus du stemming, supprime les stopwords qui sont les mots très courants de la langue anglaise n’apportant pas de valeur informative pour la compréhension du sens du texte.

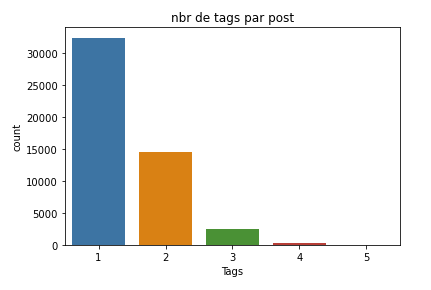
Une fois nos deux fonctions appliquées à Title et Body, nous avons traité les tags en les séparant dans des tableaux et en supprimant les ( '><' ).

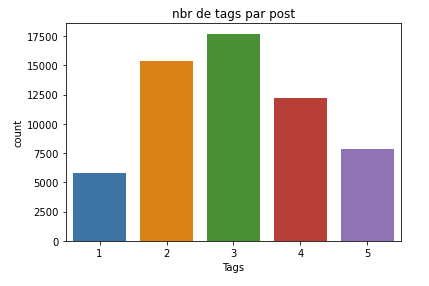
**3. Analyses univariées**

**1-Analyse et gestion des Tags**

Afin de travailler sur des tags suffisamment présents dans le dataset, nous conservons les tags apparaissant plus de 500 fois, soit 47 tags au total. Nous avons ensuite supprimé les 5 questions n’ayant plus de tags. L’objectif de cette opération est d’optimiser l’approche supervisée.

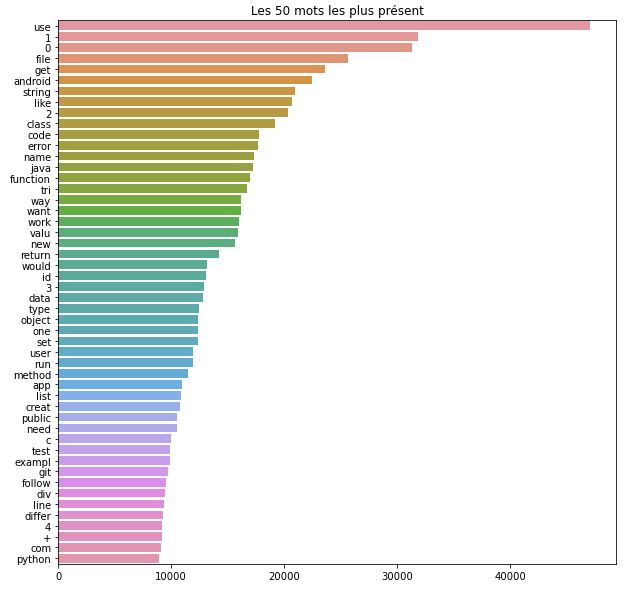
Le nombre de tags par post a changé, la plupart des questions n’ont désormais qu’un seul tag associé.





**2-Analyse de df.Document (df.Title+df.Body)**

Sur ce graphique apparait les 50 mots les plus présents dans les questions.

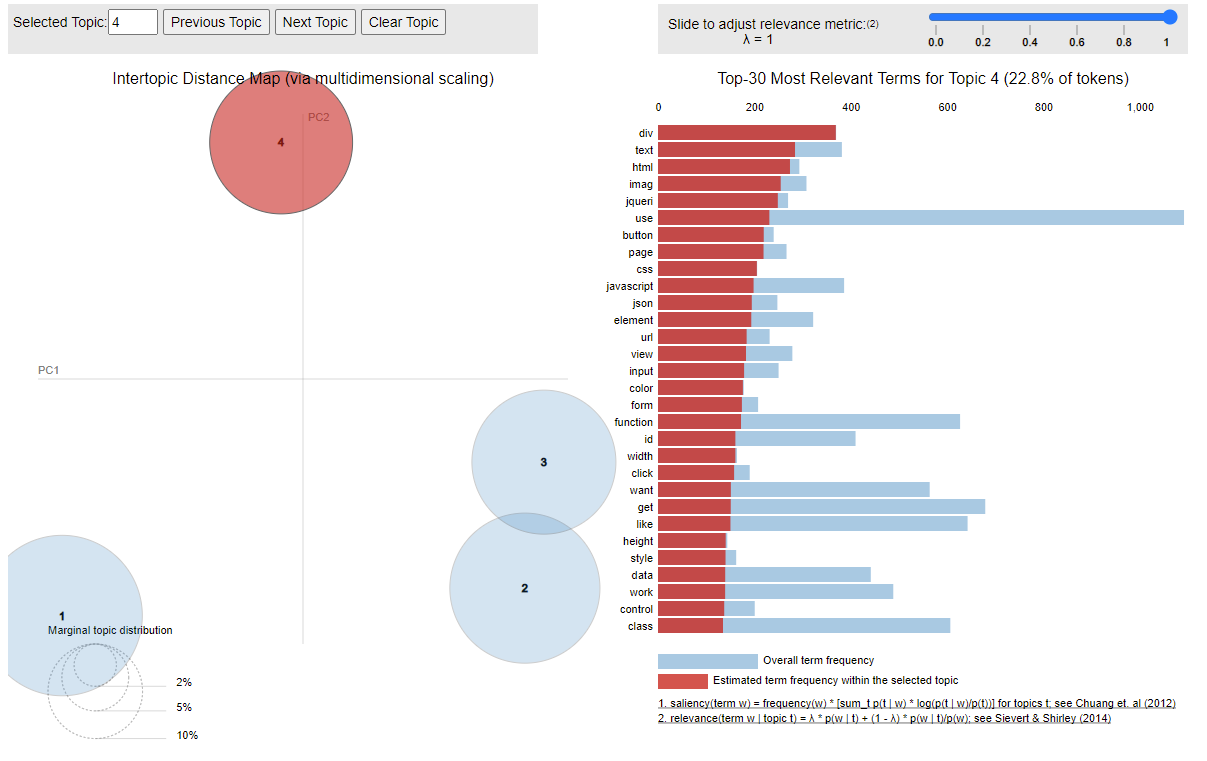
****

**3-Les modélisations effectuées**

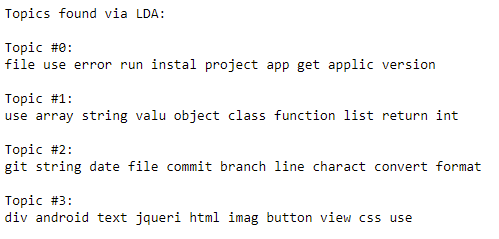
On cherche à assigner automatiquement plusieurs tags à une question. Pour cela, on va essayer deux approches différentes, une supervisée non supervisée avec le modèle LDA (Latent Dirichlet Allocation) et une approche supervisée avec différents modèles de machine learning.

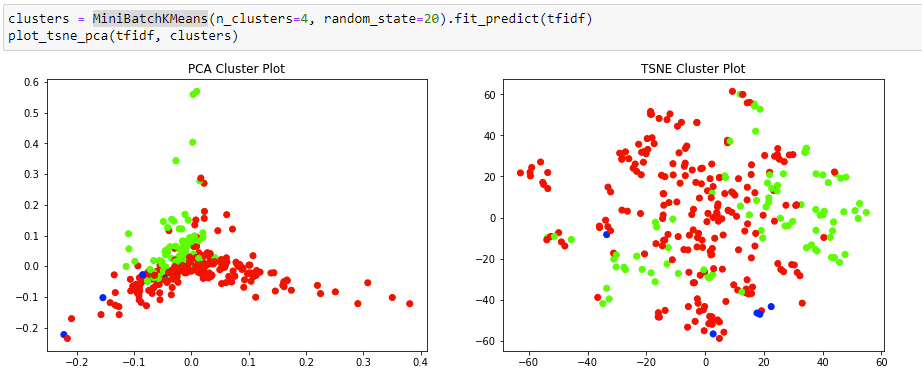
1. **Approche non supervisée**

Premièrement, on concatène les colonnes df .Title et df.Body en df.Document et on vectorise la variable df.Document avec TfidfVectorizer().

Le premier modèle étudié est un LDA (Latent Dirichlet Allocation) avec 4 « n\_components». On a observé les clusters de ce modèle avec PyLDAvis.

On affiche les 4 topics déterminés avec le LDA avec 10 mots.



On affiche 4 clusters avec un TSNE en utilisant le modèle MiniBatchKMeans().

Afin d’optimiser la modélisation du LDA, on fait une boucle sur le nombre de « n\_components » du LDA avec Sklearn, en comparants les perplexités des différents modèles et du LdaMulticore avec Gensim, en comparant la cohérence des différents modèles.

Le meilleur model avec Gensim est celui à 8 topics qu’on affiche également avec pyLDAvis.

**2-Approche supervisée**

On a effectué un multilabel\_binarizer sur df.Tags et un Tfidf Vectorizer sur df.Document et on a séparé le dataframe avec un train\_test\_split().

Les modèles que l’on a choisi d’étudier sont :

* LinearSVC(random\_state=0)
* RandomForestClassifier(random\_state=0)
* Perceptron(random\_state=0)
* LogisticRegression(random\_state=0)
* SGDClassifier(random\_state=0)
* GradientBoostingClassifier(random\_state=0)

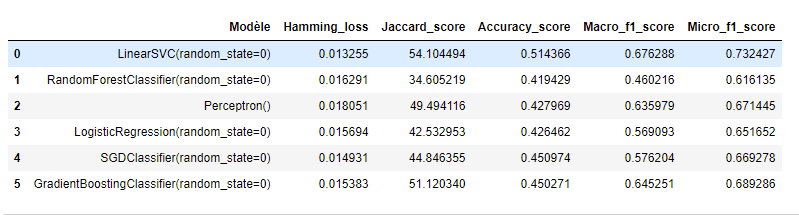
Ils sont tous ajustés grâce à un OneVsRestClassifier() avant d’être entrainés.

On a codé une fonctions « errors » pour le calcul de différentes métriques :

* Hamming\_loss : est la fraction d'étiquettes qui sont incorrectement prédites.
* Jaccard\_score : la taille de l'intersection divisée par la taille de l'union de deux jeux d'étiquettes.
* Accuracy\_score : précision

F1\_score (moyenne pondérée de la précision et du rappel) :

* Macro\_f1\_score : calcul pour chaque étiquette.
* Micro\_f1\_score : calcul global.

Voici le tableau obtenu après le calcul des différentes métriques.

**4-La modélisation choisie pour la mise en production**

Le meilleur modèle obtenu est le linear\_svc(). J’ai cherché à optimiser les hyperparamètres du model avec un GridsearchCV().

parameters = { "clf\_\_estimator\_\_C": np.arange(1,20,2),

"clf\_\_estimator\_\_max\_iter":[1000, 10000, 20000] }

Le meilleur modèle a pour hyperparamètres :

{'clf\_\_estimator\_\_C': 1, 'clf\_\_estimator\_\_max\_iter': 1000}

J’ai finalement utilisé la librairie **pickle** afin de récupérer le modèle, le tfidf\_vectorizer et le multilabel binarizer.

**5-Livrables**

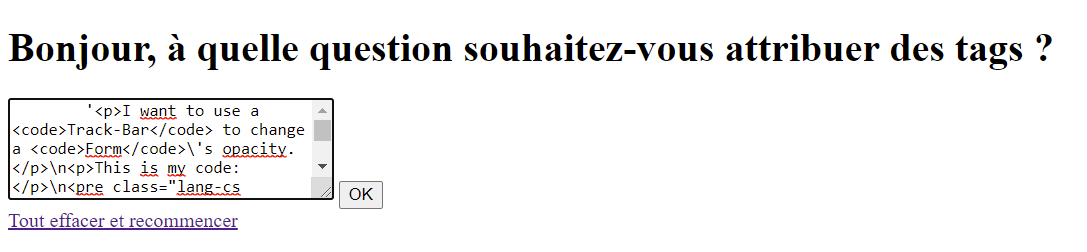
**1-Mon API**

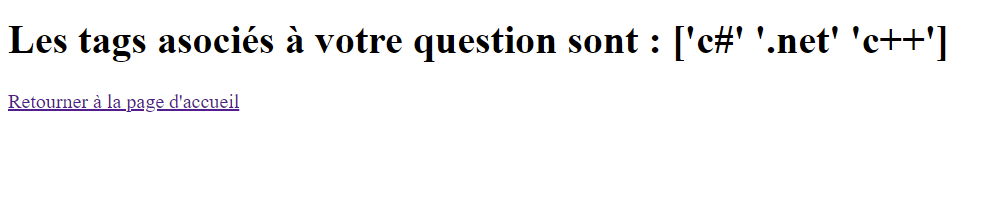
Le dossier contenant mon API est composé :

* du modèle « .pkl »
* du tfidf\_vectorizer « .pkl »
* du multilabel binarizer « .pkl »
* d’un fichier « static » contenant le code css
* d’un fichier « templates » contenant deux pages html
* d’un notebook python utilisant flask pour générer notre API en ligne.

Notre API Flask fonctionne ainsi :

* Un utilisateur entre manuellement une question
* Elle est traitée par nos 2 fonctions de tokenisation et de stemming préalablement présentées
* Elle est vectorisée
* On utilise notre modèle final pour obtenir la fonction de décision liée au texte
* On traduit cette fonction de décision avec notre multilabel\_binarizer.
* Enfin, 3 tags associés à celle-ci sont donnés automatiquement en sortie sur une nouvelle page.

Voici 2 captures d’écran des 2 pages avec un exemple de question duquel a été tiré 3 tags. Cette question est la première question présente dans mon dataset de départ.



**2-Code final déployé sur Git**

Voici le liens github sur lequel vous pourrez trouver l’ensemble des éléments de ce projet :

[**https://github.com/Thibault44/P6-TR**](https://github.com/Thibault44/P6-TR)

**6-Synthèse**

Après avoir effectué un pré-traitement de notre corpus de texte permettent de transformer le texte en features exploitables par des algorithmes de machine learning, nous avons développés et comparés, grâce à quelques métriques différentes, plusieurs modèles supervisés ou non.

L’approche la plus adaptée à notre problématique est l’approche supervisée et nous pouvons désormais déterminer les tags associés à une nouvelle question grâce à notre modèle supervisé linear\_svc(). Notre API (interface de programmation d’application) propose automatiquement 3 tags à une question.

Notre code ainsi que l’intégralité de notre projet sont retrouvables en ligne grâce à Github.