

OpenClassrooms – Parcours Ingénieur Machine Learning

# CATEGORISER AUTOMATIQUEMENT DES QUESTIONS

Projet 5 – 05 : Rapport

Florian Formentini  
03/07/2021

# Sommaire

---

I.	Étude du besoin.....	4
	Axes de recherche .....	4
II.	Analyse des données .....	5
1.	Récupération .....	5
2.	Nettoyage .....	5
3.	Analyse exploratoire .....	6
	Données temporelles.....	6
	Comptages .....	6
III.	Préparation des données .....	7
1.	Suppression des tags peu fréquents .....	7
2.	Étiquetage morpho-syntaxique .....	8
3.	Mots vides.....	9
4.	Expressions régulières .....	9
5.	Normalisation des documents .....	9
IV.	Vectorisation des questions .....	9
1.	Sac de mots.....	10
2.	TF-IDF.....	10
V.	Méthode 1 : Approche non-supervisée .....	11
1.	Topic Modeling.....	11
	Modèle LDA .....	11
	Modèle NMF.....	12
2.	Associations thèmes/tags.....	14
VI.	Méthode 2 : Approche supervisée.....	16
1.	Classifications multi-labels .....	16
	OneVsRest .....	16
	ClassifierChain .....	17
VII.	Comparaison des méthodes .....	17
VIII.	Déploiement d'une API .....	18
1.	Conception de l'application .....	18
2.	Structure de l'application .....	18
3.	Utilisation.....	19
IX.	Annexes .....	21
1.	Requête SQL .....	21

2.	Description des POS tags .....	21
3.	Sources .....	22

## I. Étude du besoin

StackOverflow, célèbre forum lié au développement informatique créé en 2008 , comporte aujourd'hui plusieurs millions de questions concernant des milliers de sujets différents. Lors de l'écriture d'une nouvelle question, il est nécessaire d'indiquer jusqu'à 5 tags pour que la question soit correctement catégorisée parmi toutes les rubriques du forum.

Cette tâche peut sembler difficile à un nouvel utilisateur qui n'est pas encore totalement familier avec le sujet abordé dans sa question ou les différentes rubriques existantes du forum. De plus, une mauvaise catégorisation nuit à l'intégrité du forum et affecte la visibilité des questions, ce qui peut décourager à en poser de nouvelles.

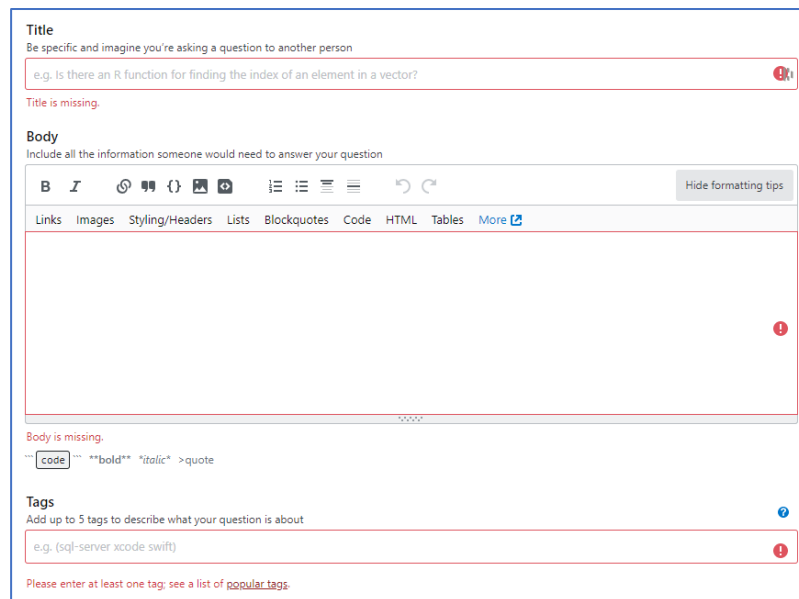


Figure 1: Formulaire de question StackOverflow

En tant que membre actif et expérimenté du site, j'ai décidé de concevoir un système de recommandations de tags pour faciliter l'étiquetage des nouvelles questions avec des tags pertinents et ainsi participer à la pérennisation de la plateforme.

## Axes de recherche

Afin de réaliser ce projet, deux approches différentes ont été envisagées :

1. La première méthode consiste à utiliser un modèle non-supervisé cherchant à détecter les thèmes principaux des questions. Et avec un ensemble de questions étiquetées il est ensuite possible d'associer les thèmes dominants détectés aux tags pour chaque questions, puis d'utiliser ces associations pour extrapoler les tags de nouvelles questions.
2. La seconde méthode se base sur l'utilisation d'un modèle supervisé de classification multi-labels entraîné sur un ensemble de questions déjà taguées et vectorisées.

Après l'évaluation de ces 2 méthodes, la plus performante a été intégrée dans une API, disponible à l'adresse <https://oc-iml-p5.herokuapp.com/>

## II. Analyse des données

---

### 1. Récupération

StackExchange, le réseau de sites dont StackOverflow fait partie, propose l'outil d'export de données [stackexchange explorer](#) qui recense un grand nombre de données authentiques de la plateforme. Avec une simple requête SQL il est donc possible de récupérer un ensemble pertinent de questions déjà taguées.

Après une étude des métadonnées de la base de données. Il a été possible d'écrire une requête SQL adaptée aux besoins du projet (cf. [Annexe 1 : Requête SQL](#)). Celle-ci permet de récupérer les champs nécessaires à la prédiction des tags ainsi que des champs additionnels permettant d'éventuels filtres supplémentaires sur la pertinence des questions. Les champs ont également des alias pour faciliter leur manipulation.

Un premier filtrage est aussi appliqué :

- Sur les types de messages pour ne récupérer que les questions (grâce à une jointure entre les tables Posts et PostTypes)
- Sur la date d'écriture pour ne récupérer que des questions récentes
- Sur les tags pour s'assurer de pas avoir de valeurs manquantes
- Sur le nombre de favoris et le score pour s'assurer qu'ils soient supérieurs à 0 en guise de premier critère de pertinence.

La plateforme limite les résultats à 50 000 lignes par requête. Il est donc possible d'ajouter des filtres supplémentaires sur l'identifiant des questions afin d'obtenir plus de données sans avoir de doublons. Cependant une seule requête a été utilisée, essentiellement pour des raisons de ressources sur la machine utilisée.

### 2. Nettoyage

Plusieurs opérations ont été effectuées pour nettoyer les données récupérées :

- Correction des types : conversion des colonnes contenant des dates
- Suppression des tags HTML dans le corps des questions
- Concaténation du titre et du corps des questions
- Passage du texte en minuscule
- Formatage des colonnes contenant les tags
- Vérification des valeurs nulles

Deux nouvelles colonnes contenant le comptage des tags et des mots (sans traitements préalable) ont également été ajoutées.

### 3. Analyse exploratoire

#### Données temporelles

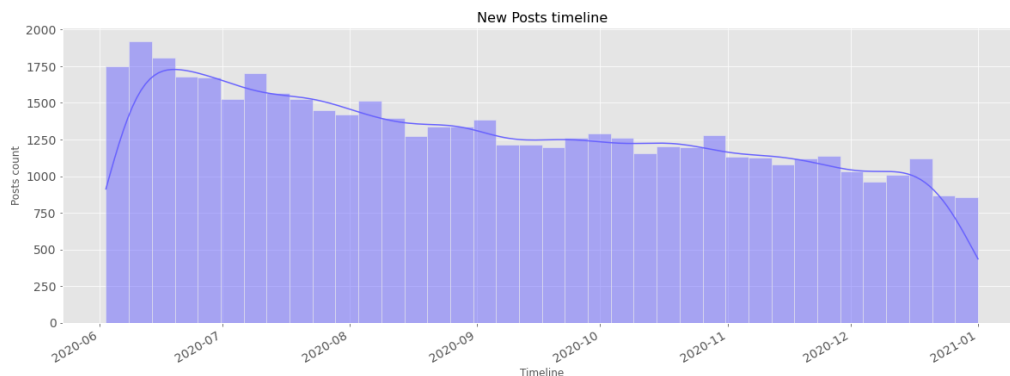


Figure 2 : Création des questions

Le jeu de données récupéré contient des questions créées **entre le 06/02/2020 et le 12/31/2020**, avec environ 1613 questions par mois. On peut aussi remarquer que le nombre de nouvelles question est décroissant.

#### Comptages

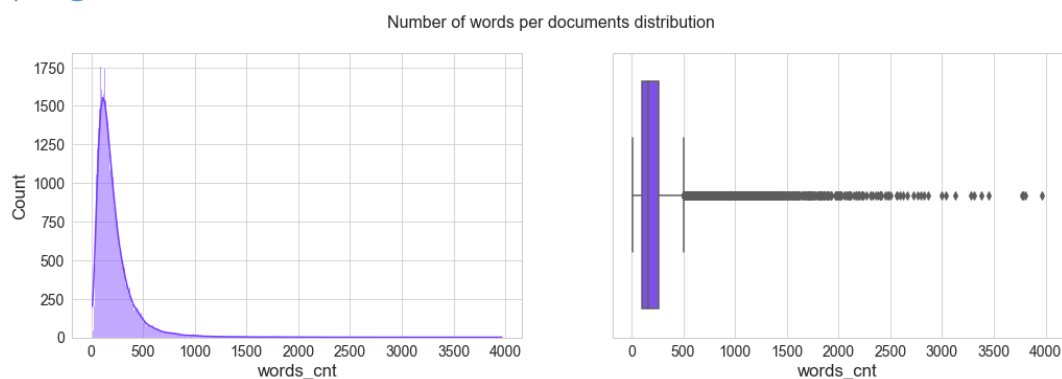


Figure 3: Répartition du nombre de tags par questions

→ 75% des questions possèdent entre 100 et 250 mots mais certaines questions sont beaucoup plus longues, le maximum étant un peu moins de 4000 mots.

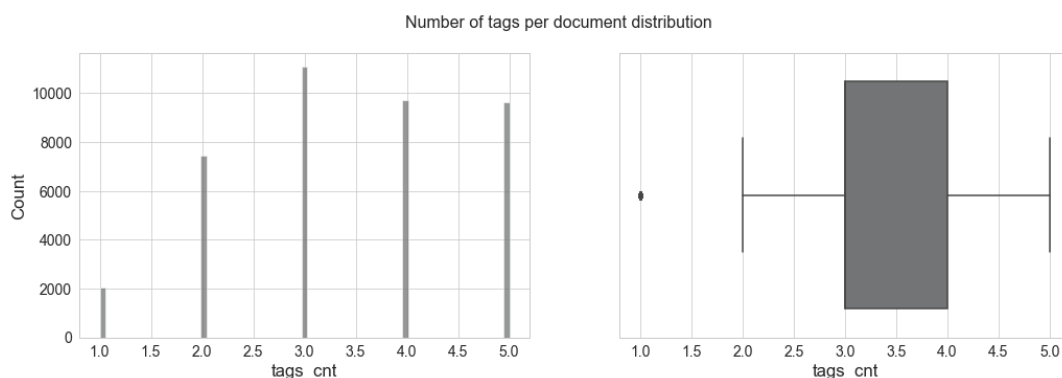


Figure 4: Répartition du nombre de tags par questions

- 75% des questions sont associées à 3 ou 4 tags et seulement peu de question n'en possèdent qu'un seul.

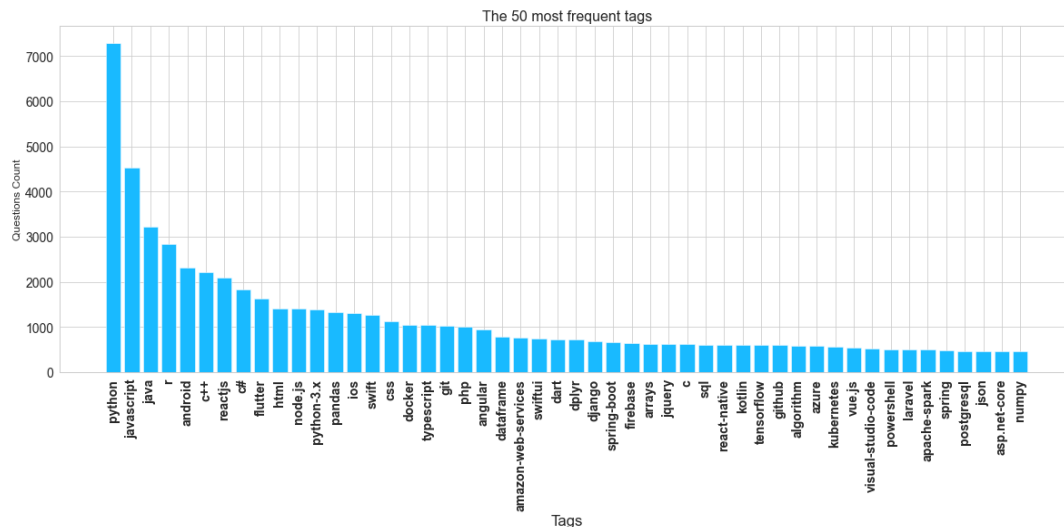


Figure 5: Les 50 tags les plus utilisés

- On peut remarquer une grande différence entre la fréquence des premiers tags et celle des suivants. Parmi les **14618 tags existants** :
- 6452 tags n'apparaissent qu'une seule fois (env. 44% des tags)
  - 207 tags apparaissent plus de 100 fois (env. 1,4% des tags)
  - Seulement 20 tags apparaissent plus de 1000 fois (env. 0,001% des tags)

### III. Préparation des données

Quelques opérations de préparation sur les données ont été nécessaires avant de pouvoir prédire les tags de chaque question.

#### 1. Suppression des tags peu fréquents

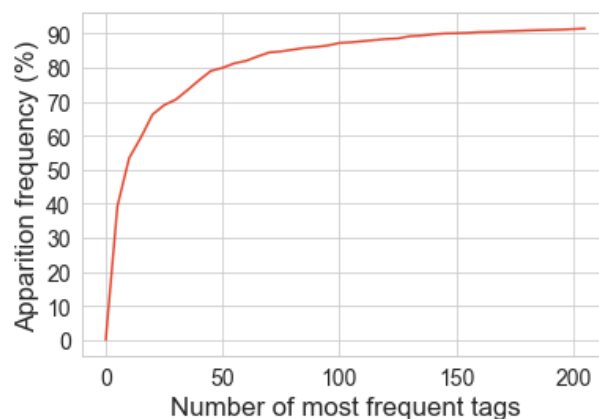


Figure 6: Fréquence d'apparitions par nombre de tags les plus fréquents

- En visualisant la fréquence cumulée des tags (triés par fréquences) il est possible de voir que les 50 tags les plus fréquents apparaissent dans environ 80% des questions.

Pour s'assurer d'avoir suffisamment de questions par tags (également pour des raisons de mémoire sur la machine utilisée), tous les autres tags ont donc été supprimés, ainsi que les questions se retrouvant désormais sans tags.

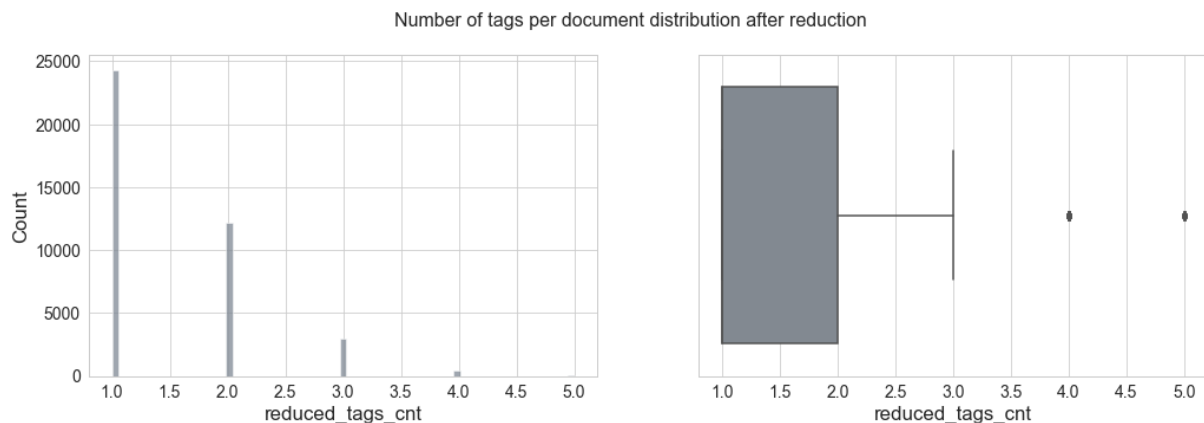


Figure 7: Répartition du nombre de tags après réduction

- Après cette réduction des tags, la majorité des questions possèdent 1 ou 2 tags et beaucoup moins qu'auparavant en ont 3 ou plus.
- Il reste **39944 questions**.

## 2. Étiquetage morpho-syntaxique

L'étiquetage morpho-syntaxique ("**Part-Of-Speech tagging**" en anglais) est l'opération consistant à associer chaque mots d'un texte à ses informations grammaticales (POS tag). Cet étiquetage peut se faire de manière très simplifiée grâce à des outils adaptés et les tags obtenus peuvent différer selon ces outils et la langue du texte.

cf. [Annexe 2 : Description des POS tags](#)

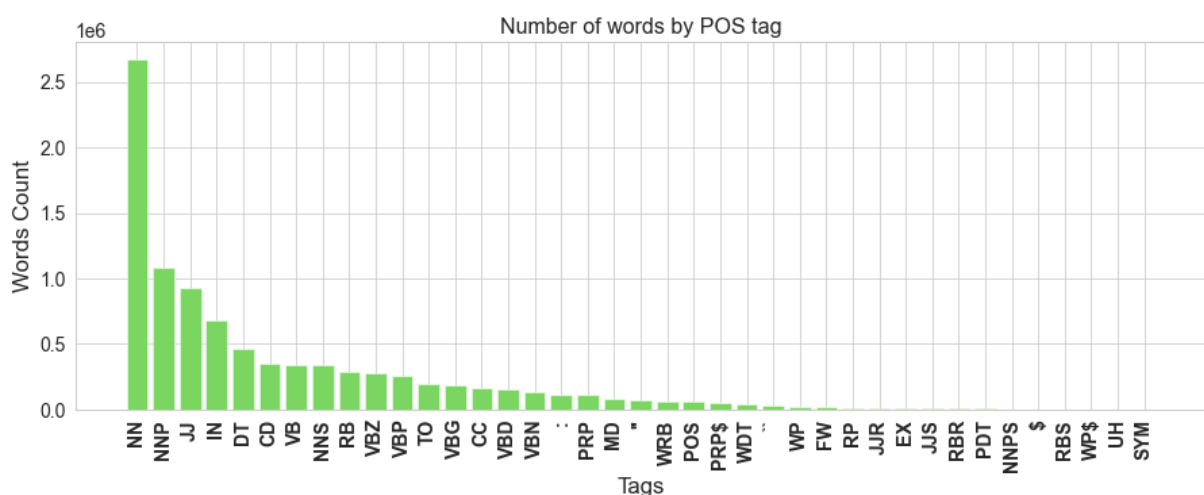


Figure 8: Répartition des POS tags

- On peut voir que la majorité des mots sont des noms mais il y a aussi de nombreux autres types de mots et des caractères seuls qui ne sont pas forcément utiles pour comprendre le contexte des documents.

Certains types de mots peuvent apporter plus d'informations que d'autres en fonction ce qu'on l'on cherche à faire. En l'occurrence, pour comprendre les sujets principaux d'une question, ce sont les



noms qui apportent le plus de renseignements, certains peuvent même être les tags que l'on cherche à découvrir. Un tri est donc fait lors de la vectorisation des questions pour **conserver seulement les noms**.

### 3. Mots vides

En recherche d'information, un mot vide (ou "**stop word**", en anglais) est un mot tellement commun qu'il est inutile de l'indexer ou de l'utiliser dans une recherche. La librairie utilisée lors de ces analyses NLTK) possède une liste de mots vides pour chaque langue supportées. Ici, la liste de mots vides de la langue anglaise est utilisée et elle comporte **179 mots**.

Après une rapide exploration de cette liste, il a été remarqué que la plupart des mots sont déjà filtrés par l'analyse morpho-syntaxique. Cependant elle est tout de même utilisée pour une double vérification et elle pourra éventuellement être étendue avec d'autre mots qui peuvent ajouter un biais aux prédictions, même si aucune analyse n'a été effectuée en ce sens pour le moment.

### 4. Expressions régulières

Les expressions régulières (aussi appelées "**regex**") sont des chaînes de caractères qui décrivent, avec une syntaxe précise, un ensemble de chaînes de caractères possibles.

Ici, une expression régulière a été utilisée pour ne conserver que les mot de 2 caractères alphanumériques ou plus et ignorer la ponctuation : `r"(?u)\b\w\w+\b"`

### 5. Normalisation des documents

Il existe plusieurs types de normalisation de documents texte. Celle utilisée dans ce cas s'appelle la **lemmatisation** et désigne un traitement lexical qui consiste à appliquer aux mots un codage renvoyant à leur forme canonique, que l'on désigne sous le terme de *lemme*. En français cela correspond à l'infinitif pour les verbes et le masculin singulier pour les adjectifs.



Figure 9: Exemple de lemmatisation en anglais

## IV. Vectorisation des questions

---

Il n'est pas possible de passer les questions telles-quelles aux modèles de Machine Learning utilisables. Il faut déjà passer par une étape de vectorisation afin de transformer le corpus en une matrice de vecteurs interprétables par les modèles, et pour cela plusieurs méthodes ont été expérimentées.

## 1. Sac de mots

La représentation par sac de mots (ou "**bag of words**" en anglais) est une méthode de vectorisation d'un corpus de documents textes qui consiste à représenter chaque document par un vecteur de la taille du vocabulaire  $|V|$  du corpus, dont la composante  $i$  indique le nombre d'occurrences du  $i$ -ème mot du dictionnaire dans ce document. En faisant cela sur le totalité du corpus on obtient une matrice termes/documents.

	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

Figure 10: Exemple de matrice Bag-Of-Words sur un corpus de 4 documents

La constitution du vocabulaire est ainsi une étape critique pour les performances des systèmes utilisant une telle représentation. Il y a donc d'autres paramètres importants en plus des opérations précédentes :

- **L'utilisation de n-grammes** : (sous-séquences de  $n$  éléments construites à partir d'une séquence donnée, ici un document).
  - Bien qu'utiliser des n-grammes peut apporter beaucoup d'informations dans certains cas (accords, conjugaison, ...), ici l'enchaînement des mots n'a pas une grande importance pour chercher les sujets principaux du corpus de questions et va affecter les performances (temps, mémoire) des modèles utilisés. Donc le vocabulaire ne **contiendra que des mots seuls** (ou *uni-grammes*)
- **Les seuils de fréquence minimale et maximale des mots à conserver** : Cela permet d'ignorer les mots avec une fréquence d'apparition trop haute ou trop faible.
  - Pour ces deux paramètres, plusieurs valeurs ont été testées, le but étant de ne pas avoir un vocabulaire trop long pour des raisons de mémoire, sans éliminer des mots potentiellement utiles
  - Finalement, en considérant que les mots en rapport avec les sujets du document sont des mots qui apparaissent plusieurs fois dans ce document sans être trop fréquent non plus, il a été décidé de filtrer les mots apparaissant **moins de 5 fois** dans un documents et les mots dont la **fréquence est supérieure à 75%** dans un même document.

**Le vocabulaire ainsi obtenu comporte 18104 mots.**

## 2. TF-IDF

Cette deuxième méthode de vectorisation consiste à pondérer la fréquence d'apparition de chaque terme par un indicateur de similarité entre tous les documents, ce qui permet de prendre en compte si ce terme est commun ou rare dans les autres documents.

La méthode de pondération TF-IDF utilise l'*inverse document frequency* (idf) qui est l'inverse de la proportion de document qui contient le terme à l'échelle logarithmique. Et cette métrique permet donc d'évaluer l'importance d'un terme contenu dans un document, relativement aux corpus de questions.

$$\text{tfidf}_{ij} = \text{tf}_{ij} \times \log \frac{|D|}{|\{d_j: t_i \in d_j\}|}$$

- $tf_{ij}$  : fréquence du terme  $t_i$  dans le document  $d_j$
- $|D|$  : nombre total de documents dans le corpus
- $|\{d_j: t_i \in d_j\}|$  : nombre de documents où le terme  $t_i$  apparaît.

De cette manière, le poids augmente proportionnellement au nombre d'occurrences du terme dans le document et varie également en fonction de la fréquence du terme dans le corpus.

En ce qui concerne la création du vocabulaire, les mêmes traitements que pour le Bag-Of-Words ont été utilisés.

## V. Méthode 1 : Approche non-supervisée

---

Cette première méthode s'appuie sur l'utilisation d'un modèle non-supervisé permettant de détecter les thèmes latents abordés dans un corpus de documents et assigner les thèmes détectés à ces différents documents (ce qui s'appelle *topic modeling* en anglais).

En appliquant cela à un corpus de document dont les tags sont connus, il est possible d'associer les thèmes dominants détectés aux tags des questions, puis utiliser ces associations pour déduire les tags d'une nouvelle question après détection de ses thèmes.

### 1. Topic Modeling

Deux modèles ont été testés : LDA et NMF. Pour évaluer les performances de ce genre de modèle il est possible d'utiliser un **score de cohérence** qui mesure la distance relative entre les mots au sein d'un thème et en retourne la moyenne. Il existe plusieurs métriques de ce genre et celle utilisée ici est **C\_V** qui retourne une valeur en 0 et 1. Avec 0 signifiant que les mots mesurés n'ont rien de semblable et 1 qu'ils sont identiques.

D'après les recherches effectuées, **ce score peut être considéré bon à partir d'environ 0.55 et s'il tend vers 0.7**. Un score plus bas signifie que les mots associées à un thème sont très peu semblables et peut signifier que le paramétrage du modèle n'est pas le bon (nombre de thèmes,  $\alpha$ ,  $\beta$ ) ou un problème avec les données. Un score plus haut est peu probable mais traduit sûrement un problème également.

### Modèle LDA

Le modèle LDA, pour *Latent Dirichlet Allocation*, est une méthode non-supervisée générative permettant d'expliquer des ensembles de documents (les questions) par le moyen de groupes non observés (les thèmes), eux-mêmes définis par des similarités de données.

Si les observations sont les mots collectés dans un corpus de documents textuels, le modèle LDA suppose que chaque document est un mélange d'un petit nombre de thèmes, et que la génération de chaque occurrence d'un mot est attribuable à l'un des thèmes du document.

On fixe alors un nombre de thèmes et on cherche à apprendre les thèmes représentés dans chaque document et les mots associés à ces thèmes en attribuant un thème à chaque mot de chaque document, selon une distribution de Dirichlet.

L'un des paramètres les plus importants pour ce genre de modèle est donc **le nombre de thèmes recherchés**. L'optimisation de ce paramètre s'est faite de manière itérative de 2 à 50 thèmes (nombre total de tags dans le jeu de données) en entrainant un modèle puis en calculant son score de cohérence.

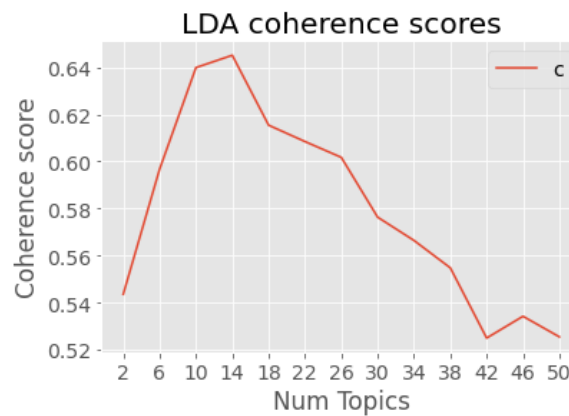


Figure 11: Optimisation du nombre de thèmes (LDA)

Comme on peut le voir, le meilleur score est obtenu avec **14 thèmes (0.645)**.



Figure 12: Les 10 mots les plus représentatifs des thèmes (LDA)

## Modèle NMF

Le modèle NMF, pour *Non-Negative Matrix Factorisation*, est une méthode statistique pour réduire la dimension des corpus d'entrée, qui utilise la méthode d'analyse factorielle pour fournir comparativement moins de poids aux mots avec moins de cohérence.

En considérant une matrice  $V(m \times n)$  dont toutes les valeurs sont positives, cette méthode permet de factoriser  $V$  en deux matrices  $W(m \times k)$  et  $H(n \times k)$  en supposant qu'elles sont toutes les 2 positives étant donné que toutes les entrées de  $V$  sont positives.

$$\begin{bmatrix} W \end{bmatrix} \times \begin{bmatrix} H \end{bmatrix} \approx \begin{bmatrix} V \end{bmatrix}$$

Figure 13: Représentation du modèle NMF

Dans le cas présent,  $V$  correspond à la matrice termes/documents obtenue par empilement des bag-of-words, la matrice  $W$  contient les poids d'appartenance des  $m$  documents aux  $k$  thèmes et la matrice  $H$  contient les poids d'appartenance des  $n$  termes aux  $k$  thèmes.

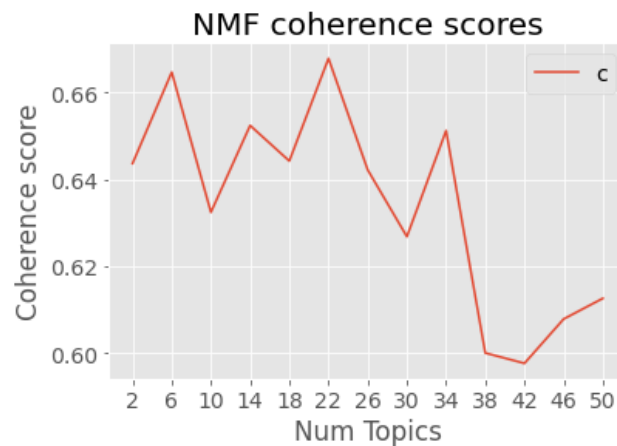


Figure 14 : Optimisation du nombre de thèmes (NMF)

L'optimisation du nombre du thèmes montre que le meilleur score de cohérence est obtenu avec **22 thèmes (0.668)**.

Le score obtenu étant également supérieur au meilleur score du modèle LDA, **c'est donc le modèle NMF qui a été retenu pour poursuivre cette méthode.**



Figure 15: Les 10 mots les plus représentatifs des thèmes (NMF)

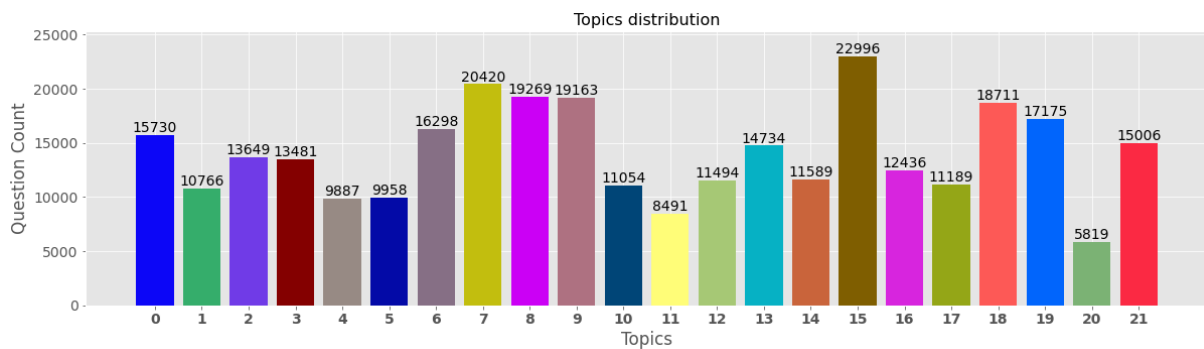


Figure 16: Distribution des thèmes détectés

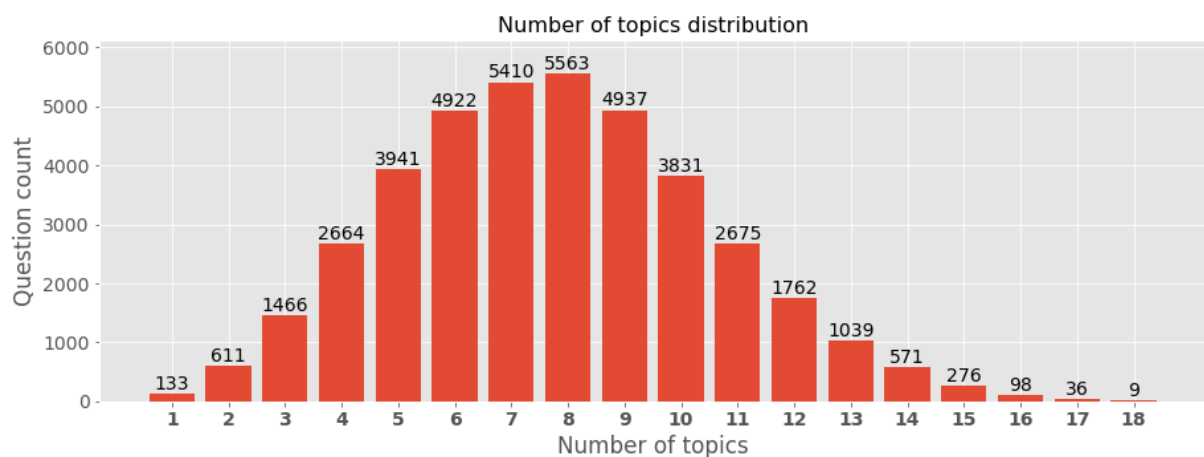


Figure 17: Distribution du nombre de thèmes détectés par questions

- En étudiant les nuages de mots, on peut identifier certains termes identiques à des tags connus. Cependant les autres thèmes comportent des mots communs à plusieurs tags, il donc plus difficile de trouver une correspondance dans les tags connus.
- Les questions sont le plus souvent associées à plusieurs thèmes, 18 au maximum, mais une majorité de questions en possèdent 8.

## 2. Associations thèmes/tags

Comme les tags de ses documents sont connus, on dispose de la matrice  $A(\text{documents}, \text{tags})$ . Et la première étape de *Topic Modeling* permet d'obtenir une matrice  $B(\text{documents}, \text{topics})$  (la matrice  $W$  du modèle NMF).

En multipliant ces deux matrices, on obtient la matrice  $C(\text{tags}, \text{topics})$  qui, une fois normalisée par thèmes donne la probabilité de chaque tags d'appartenir à ce thème.

$$C(\text{tags}, \text{topics}) = A(\text{documents}, \text{tags})^T \times B(\text{documents}, \text{topics})$$

Et en récupérant le tags avec la plus haute probabilité pour chaque thème, il est possible de de créer un dictionnaire qui permet d'associer un document à des tags en fonction de ses thèmes principaux.

Avec les données utilisées et les 22 thèmes détectés grâce au modèle NMF, voici les associations obtenues:

- |                |            |
|----------------|------------|
| 0. javascript  | 12. python |
| 1. python      | 13. java   |
| 2. java        | 14. python |
| 3. python      | 15. python |
| 4. javascript  | 16. python |
| 5. python      | 17. python |
| 6. python      | 18. python |
| 7. python      | 19. python |
| 8. javascript  | 20. python |
| 9. python      | 21. c++    |
| 10. javascript | 22. python |

On peut voir qu'il n'y a que 4 tags différents sur les 50 connus et que ces tags sont associés à plusieurs thèmes.

Ces tags font partie des plus fréquents, ce qui peut expliquer pourquoi ce sont ceux qui ressortent, cependant cette approche n'a pas permis de détecter les tags plus rares ni ceux moyennement utilisés.

Et même parmi les plus fréquents, certains ont dû être filtrés lors de la création du dictionnaire utilisé par le modèle.

En remplaçant les thèmes détectés par les tags ainsi obtenus, puis en supprimant les doublons, on obtient la distribution de tags suivante :

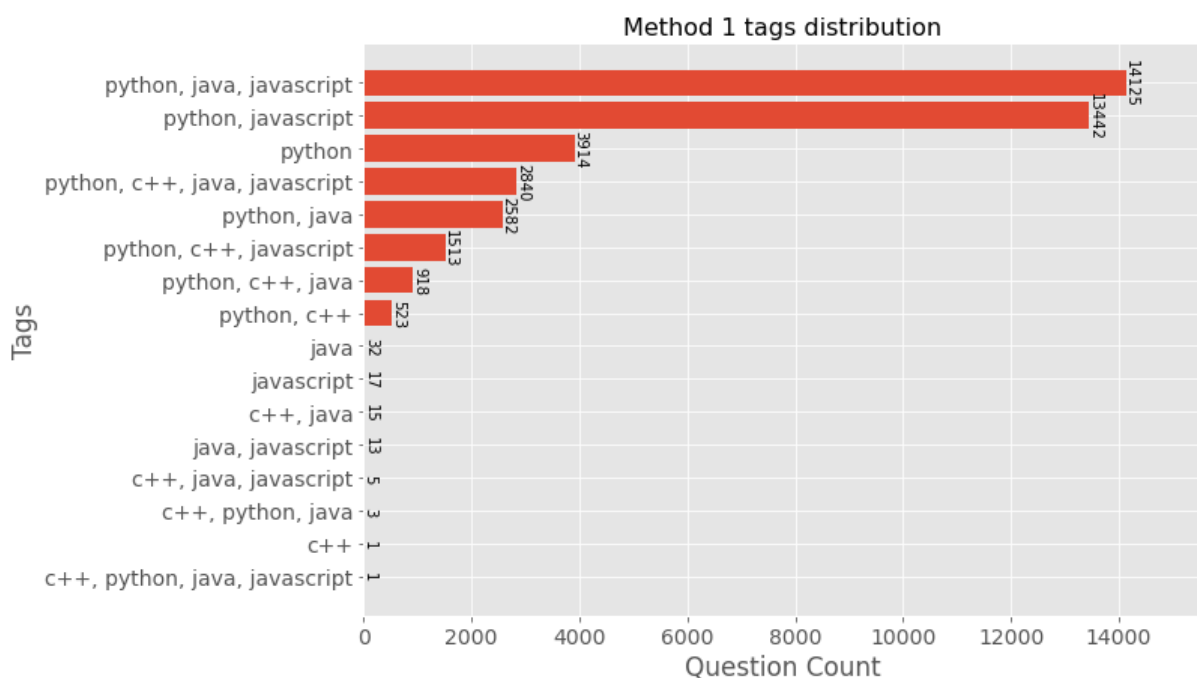


Figure 18: Distribution des tags obtenus avec l'approche non-supervisée

Même si ces combinaisons sont éventuellement possibles, cette distribution est visiblement incorrecte. En comparant les tags ainsi obtenus avec les tags réels de chaque question, il est possible de calculer le **F-score qui est de 0.025**.

**Il est donc clair que cette approche n'apporte pas de résultats satisfaisants.**

## VI. Méthode 2 : Approche supervisée

---

La seconde méthode utilise un modèle supervisé de classification multi-label afin de prédire le/les tags grâce à la matrice TFIDF obtenue après vectorisation des questions récupérées.

Avant de commencer la recherche d'un modèle adapté, quelques opérations supplémentaires de préparation à effectuer sur les données sont nécessaires :

- Séparations des données en deux. Avec **deux tiers utilisées pour l'entraînement** du modèle, et le tiers restant pour son évaluation.
- Création de matrices TFIDF pour les nouveaux jeux de données. Toutes les deux basées sur le dictionnaire du jeu d'entraînement.
- Encodage des tags sous forme de matrices binaires indiquant pour chaque question l'appartenance ou non à chacun des tags connus, également basées sur les tags du jeu d'entraînement.

### 1. Classifications multi-labels

Deux méthodes de classification ont été testées : OneVsRest et ClassifierChain. Et l'évaluation de leurs performances, s'est faite en utilisant deux métriques :

- Le F-score, qui est une mesure de précision d'un test (moyenne harmonique de la précision et du rappel du test) :

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

- L'indice de Jaccard, utilisé pour évaluer la similarité entre les échantillons (rapport entre le cardinal de l'intersection des ensembles considérés et le cardinal de l'union des ensembles)

$$J(S_1, S_2, \dots, S_N) = \frac{|S_1 \cap S_2 \cap \dots \cap S_N|}{|S_1 \cup S_2 \cup \dots \cup S_N|}$$

### OneVsRest

OneVsRest est une méthode de classification multi-classes qui consiste à appliquer un modèle de classification binaire par classe (= tag) avec chaque classifieur comparant leur classe à toutes les autres. En plus de son efficacité au niveau du temps de calculs, il est possible d'obtenir des informations sur la qualité des prédictions pour chaque classe en regardant le classifieur correspondant.

Pour obtenir des performances optimales, le choix du paramétrage s'est fait grâce à une recherche sur grille (**GridSearch**) adaptée en fonction du classifieur binaire utilisé et 3 classifieurs différents ont été testés : SVM, régression logistique et forêt aléatoire.

Ces modèles ont obtenus des résultats très proches, sauf la forêt aléatoire dont les performances sont inférieures.



## ClassifierChain

La méthode de classification multi-labels ClassifierChains applique également un classificateur binaire par classe, mais contrairement au OneVsRest, les classifieurs sont ensuite organisés en une chaîne. Chaque classifieur est alors ajusté sur les données d'apprentissage disponibles plus les véritables étiquettes des classes des modèles précédents et les prédictions sont transmises aux modèles suivants de la chaîne.

En revanche ce modèle nécessite beaucoup plus de temps de calcul que le précédent, ce qui a été un frein pour l'optimisation du modèle. En utilisant le classifieur binaire ayant retourné les meilleurs scores avec la méthode précédente (et les paramètres trouvés lors de son optimisation), ce modèle a pris plus de 7h à s'entraîner pour finalement obtenir un score à peu près similaire mais légèrement inférieur au modèle OneVsRest et régression logistique.

## VII. Comparaison des méthodes

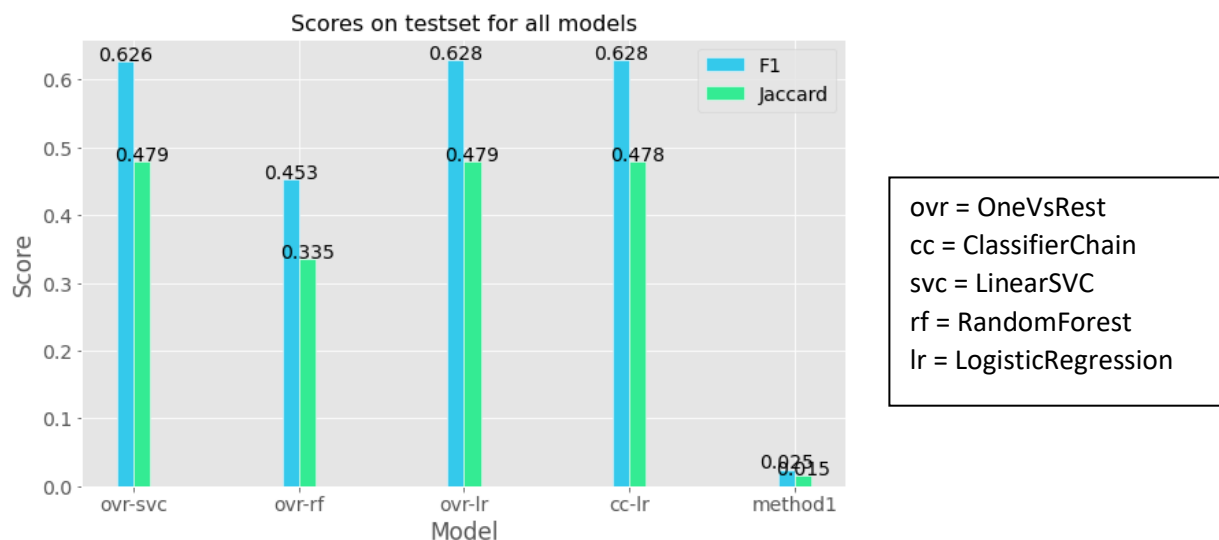


Figure 19: Scores des modèles testés

Comme on peut le voir, les modèles de l'approche supervisée ont produits de bien meilleurs résultats que l'approche non-supervisée.

Les modèles de l'approche supervisée ont des résultats à peu près similaires entre eux, sauf pour le modèle utilisant la forêt aléatoire dont les scores nettement inférieurs.

La différence de performance entre les méthodes de classification OneVsRest et ClassifierChain ne permet pas d'affirmer qu'une des méthodes est meilleure que l'autre pour le problème étudié, et en raison du temps de calcul nécessaire, les tests n'ont pas été continués.

Le modèle choisi pour poursuivre le projet est donc le modèle supervisée utilisant la méthode **OneVsRest et une régression logistique** en classifieur binaire car c'est celui ayant les meilleures performances. Le paramétrage utilisé est le suivant :

- **solver** : 'saga', qui d'après la documentation est plus efficace sur de grands datasets
- **dual** : False, car il y a plus d'échantillons que de features.
- **C** : 10, trouvé grâce à la recherche sur grille
- **penalty** : 'l1', trouvé grâce à la recherche sur grille

## VIII. Déploiement d'une API

### 1. Conception de l'application

L'application se base sur le **micro-framework Flask**. Celui-ci a pour objectif de garder un noyau simple mais extensible. Il n'intègre pas de système d'authentification, pas de couche d'abstraction de base de données, ni d'outil de validation de formulaires mais de nombreuses extensions permettent d'ajouter facilement ces fonctionnalités.

Plusieurs librairies ont donc été utilisés, notamment **Flask-RestX**, qui permet une création facilitée d'API REST. Flask apporte aussi quelques concepts pratiques comme les **Blueprints** qui permettent d'organiser le projet en groupant des fonctionnalités dans des sous-dossiers. Ce qui peut s'associer au concept de **Namespaces** de Flask-RestX rendant possible un regroupement de fonctionnalités sur des routes préfixées.

Cela permet de créer des routes sur le modèle suivant : **<host>/namespace/endpoint[/params]**

- **<host>** : L'adresse de l'hébergeur. Un blueprint 'api' avec une **documentation Swagger** est directement disponible sans préfix car aucune page d'accueil n'est prévue.
- **/namespace** : Le préfix d'un Namespace, en l'occurrence **'tagsuggestion'**
- **/endpoint[/params]** : La route désirée et ses potentiels paramètres

De plus, l'implémentation du **pattern factory** de Flask permet de faire tourner plusieurs instances de l'application indépendamment, ce qui permet par exemple un versionning de l'API ou de faire tourner plusieurs environnements (dev/test/prod) simultanément.

Les différentes routes sont sécurisées avec une clé d'API qu'il est nécessaire de passer dans l'en-tête des requêtes lors des appels, dans le champs **X-API-KEY**, afin de pouvoir utiliser les fonctionnalités de l'application. Cette clé secrète est stockée en tant que variable d'environnement de l'hébergeur pour qu'elle n'apparaisse pas directement dans les fichiers de l'application (et sur GitHub).

Enfin, le projet a été **déployé sur l'hébergeur Heroku** avec le déploiement automatique est configuré, c'est-à-dire qu'à chaque **push** sur la branche master du projet, l'API sera redéployée (uniquement le dossier de l'API). L'application est disponible à l'adresse suivante : <https://oc-iml-p5.herokuapp.com/>

### 2. Structure de l'application

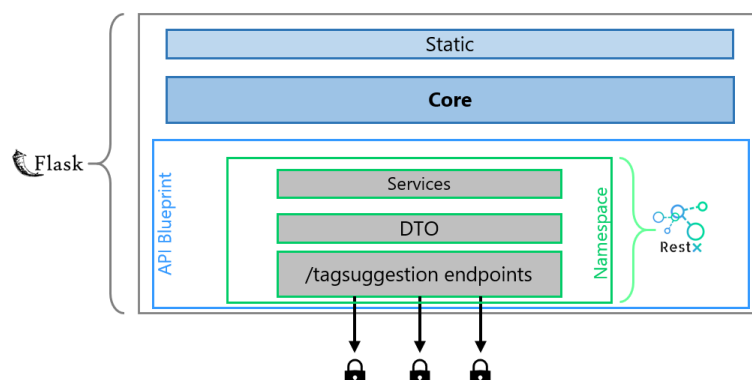


Figure 20: Structure globale de l'application

- **Dossier static** : contient des fichiers statiques nécessaires à l'application (modèles utilisés)
- **Module core** : contient la logique métier (traitement du texte et prédictions)
- **Package API** : Blueprint contenant l'API
- **Package tagsuggestion** : Namespace contenant les modules de l'API
- **Module services** : contient une classe permettant de traiter les requêtes entrantes et les réponses du namespace.
- **Module DTO** : contient la déclaration de modèles de validation des données qui entrent/sortent ainsi que des paramètres possibles pour les routes du namespace.
- **Module endpoints** : contient la déclaration des routes et leur documentation Swagger
- Le namespace api contient également un module **"utils"** permettant de gérer la sécurisation des routes et la gestion d'envois de fichiers

### 3. Utilisation

Se rendre à l'adresse <https://oc-impl-p5.herokuapp.com/> et dérouler les routes du Namespace *Tags Suggestion*.

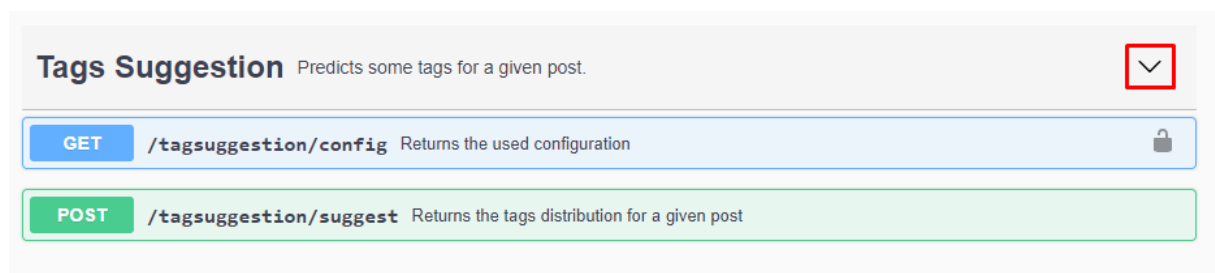


Figure 21: Routes du Namespace Tags Suggestion

En envoyant une requête POST qui contient une question à la route `/tagsuggestion/suggest`, la réponse contiendra les tags prédits. Un paramètre booléen *return-post* permet également de décider la réponse doit contenir la question reçue et son titre.

Il est possible de tester le route directement depuis l'interface Swagger en cliquant sur celle-ci, puis sur le bouton "Try it out". Il sera alors possible d'indiquer un titre et une question dans l'encadré affiché et de cliquer sur le bouton "Execute" pour envoyer la requête.

Si le bouton est grisé cela signifie qu'il y a une erreur de syntaxe.

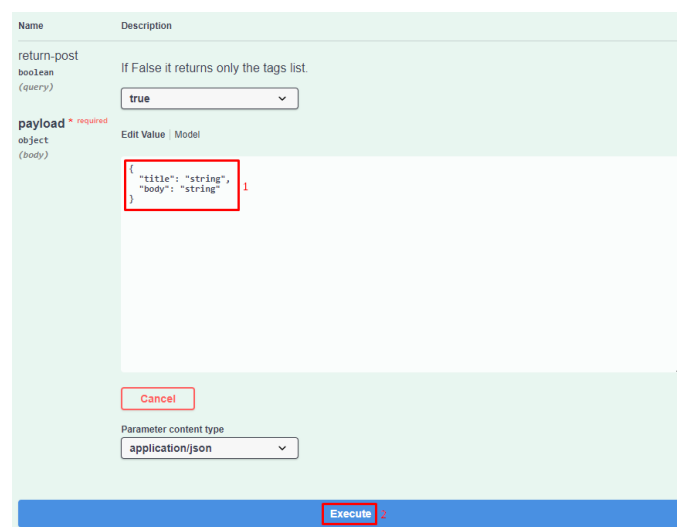


Figure 22: Test de la route `/tagsuggestion/suggest`

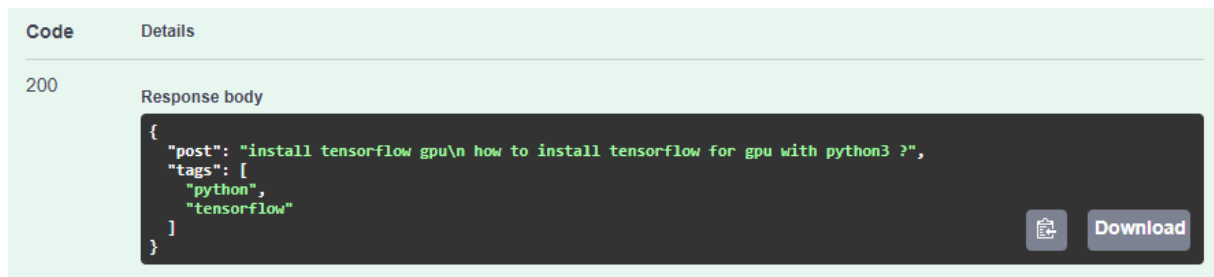


Figure 23: Prédiction de tags

Il existe également une seconde route : `/tagsuggestion/config` qui retourne la configuration du modèle utilisé. Cependant il faut utiliser la clé d'API **dev\_apikey** dans la requête pour l'utiliser. L'interface Swagger permet également de le faire avec le bouton "Authorize" en haut à droite de la page.

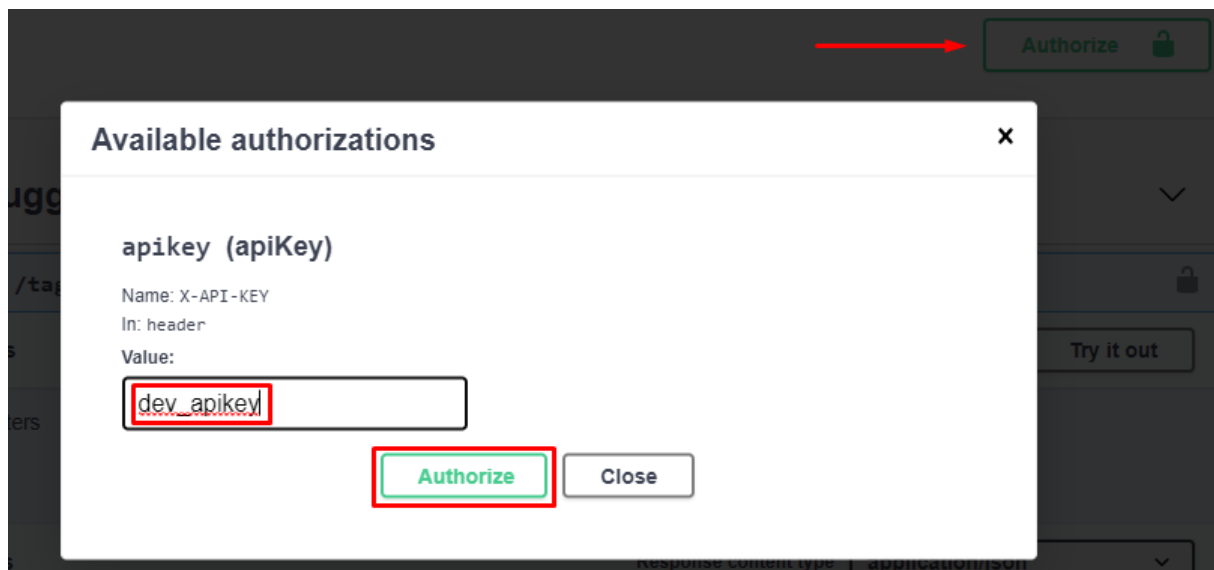


Figure 24: Authentification sur Swagger

En ce qui concerne l'installation et l'utilisation de l'application en local, la procédure est détaillée dans le fichier [README du dossier de l'API](#), visible sur Github.

## IX. Annexes

### 1. Requête SQL

Requête SQL utilisée pour récupérer les données depuis StackExchange.

```
SELECT
  p.Id as id,
  p.Body as doc,
  p.Title as title,
  p.Tags as tags,
  p.CreationDate as creation_date,
  -- some additionnal metrics
  p.Score as score,
  p.ViewCount as views,
  p.AnswerCount as answers,
  p.CommentCount as comments,
  p.FavoriteCount as favorites,
  p.LastActivityDate as last_activity_date
FROM posts p, PostTypes pt
WHERE p.PostTypeId = pt.Id           -- join between Post and PostTypes
AND pt.Name = 'Question'           -- to get only questions
AND p.Tags IS NOT NULL              -- filter questions without tags
AND p.FavoriteCount > 0              -- to get relevant questions
AND p.Score > 0
AND p.CreationDate <= '01-01-2021' -- to get recent questions
-- to do several queries without duplicates
[AND p.Id >= XXX
AND p.Id < XXX]
```

### 2. Description des POS tags

Les tags obtenus dépendent du corpus utilisé pour entrainer le tagueur. Dans le cadre de ce projet, la détection des tags s'est faite avec la librairie NLTK qui utilise le jeu de tags du "[Penn Treebank Project](#)". Voici la description de ces tags :

N°	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative

10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP\$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP\$	Possessive wh-pronoun
36.	WRB	Wh-adverb

### 3. Sources

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>– <a href="#">Définition Stop words</a></li> <li>– <a href="#">Définition Lemmatisation</a></li> <li>– <a href="#">Définition Part of Speech Tagging</a></li> <li>– <a href="#">Penn Treebank POS tags</a></li> <li>– <a href="#">Définition d'un bag of words</a></li> <li>– <a href="#">Word Embedding</a></li> <li>– <a href="#">Universal Sentence Encoder</a></li> <li>– <a href="#">Définition TFIDF</a></li> </ul> | <ul style="list-style-type: none"> <li>– <a href="#">Définition LDA</a></li> <li>– <a href="#">Définition NMF</a></li> <li>– <a href="#">Définition du score de cohérence</a></li> <li>– <a href="#">Définition de l'indice de Jaccard</a></li> <li>– <a href="#">Définition OneVsRest</a></li> <li>– <a href="#">Définition ClassifierChain</a></li> <li>– <a href="#">Documentation Flask</a></li> <li>– <a href="#">Documentation Flask-RestX</a></li> </ul> |
|--|---|