

# Création d'un système de suggestion de tags pour Stack Overflow

Manon Giraud

## Table des matières

Introduction.....	3
Contexte et problématique .....	3
Récupération des données .....	3
Traitement des données .....	4
Analyse non supervisée.....	6
LDA .....	6
NMF.....	7
Analyse supervisée .....	8
Multiclass ou multilabel .....	8
SVC .....	9
Forêts aléatoires .....	9
Choix du seuil.....	10
Résultats .....	10
API.....	11
Conclusion .....	11

# Introduction

## Contexte et problématique

Le site Stack Overflow permet aux utilisateurs de soumettre des questions à d'autres internautes et d'y répondre. Vu l'énorme quantité de thèmes de questions possibles, des tags sont utilisés pour indiquer le sujet de la question, et faciliter l'usage du moteur de recherche.

Il peut cependant être difficile pour un utilisateur de savoir quels seraient les tags les plus appropriés pour sa question, surtout s'il a peu l'habitude d'utiliser le site.

Le but de ce projet est donc de suggérer des tags appropriés en fonction de la question et de son titre, grâce au traitement automatique des langues, ou natural language processing (NLP).

## Récupération des données

Les textes utilisés pour ce projet proviennent du site [stackexchange.com](http://stackexchange.com), qui propose un outil permettant de récupérer des données de Stack Overflow à l'aide de requêtes SQL. En particulier, les questions posées sur les utilisateurs qui se trouvent dans la table Posts.

Cet outil est limité en temps d'exécution ainsi qu'en nombre de résultats disponibles simultanément : une requête qui met trop de temps à s'exécuter ne donnera pas de résultats, et un maximum de 50 000 individus pourront être obtenus simultanément.

Pour pallier à ces contraintes, j'ai inclus dans la requête des conditions sur la variable Id. Un maximum évite que la recherche ne prenne trop de temps, tandis qu'un minimum sur la deuxième requête permet d'obtenir des individus qui n'étaient pas dans les résultats de la première.

Les autres contraintes sont sur PostTypeId, pour n'obtenir que les posts de type question, et sur AnswerCount, pour restreindre les résultats aux questions pertinentes, soit celles ayant reçu au moins une réponse. Les variables sélectionnées sont Id (l'identifiant des questions), Body et Title (qui contiennent le texte utilisé dans ce NLP), et Tags (la variable que nous essayerons de prédire).

Les requêtes en question ressemblent donc au suivant :

```
SELECT ID, BODY, TITLE, TAGS FROM POSTS WHERE 396216<ID AND ID<1000000 AND ANSWERCOUNT>1 AND POSTTYPEID=1
```

Je me suis limitée à 100 000 individus, car c'est un nombre suffisant pour effectuer des analyses poussées, sans être suffisamment grand pour que le temps d'exécution des analyses devienne ingérable.

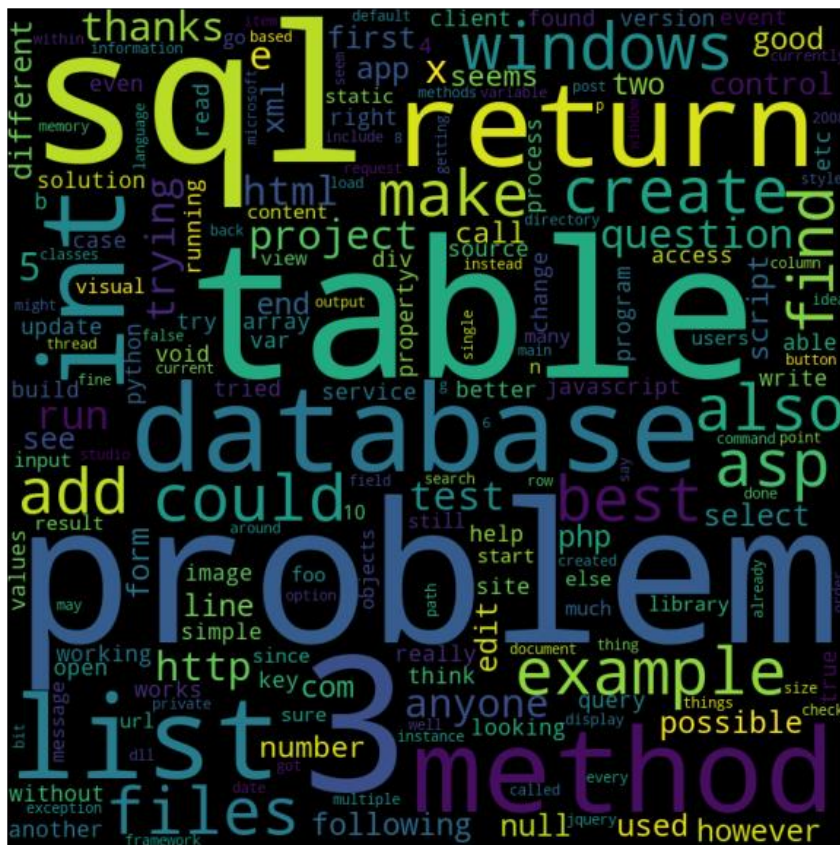
Il est à noter que les résultats obtenus sont ceux ayant un Id très petit, comparé à la totalité de la base de données. Cela signifie que les questions sont parmi les plus anciennes du site, et que les tags prédits sont donc peut-être moins utilisés actuellement. Cela étant dit, la méthode utilisée dans ce projet resterait la même quel que soit l'âge des données.

## Traitement des données

Après la récupération des données, il a d'abord fallu les transformer, pour qu'elles soient dans un format utilisable pour l'analyse.

J'ai commencé par regrouper les résultats des requêtes en une seule table, contenant 100 000 individus plutôt que 50 000, et par concaténer les contenus des variables Title et Body pour chaque individu, afin de n'avoir qu'un texte par individu. J'ai également formaté les données, en supprimant les codes HTML et la ponctuation, en mettant tous les mots en minuscule et en mettant le texte sous forme de liste de mots.

En plus de ces changements de format, il faut transformer les données plus en profondeur, car un texte brut n'est pas utilisable pour la NLP. En effet, il existe dans toutes les langues des mots qui n'apportent rien au sens du texte, et qui ne sont donc pas utiles pour la classification. Ce sont des stopwords. De plus, dans un corpus de texte donné, il y a des mots extrêmement communs, qui apparaîtront dans toutes les classes (par exemple, le mot « user »). Ces mots – les stopwords et les mots trop fréquents – ne seront pas utiles pour classer notre texte, il convient donc de tous les supprimer.



Mots les plus fréquents, après suppression des stopwords

En plus de cela, certains mots dans une langue ont le même sens, mais des différences d'orthographe. Citons par exemple les mots au singulier et au pluriel, ou la conjugaison. Séparer ces mots presque identiques réduirait la qualité de notre analyse, c'est pourquoi j'ai utilisé la racinisation, qui permet de réduire les mots à leur racine.

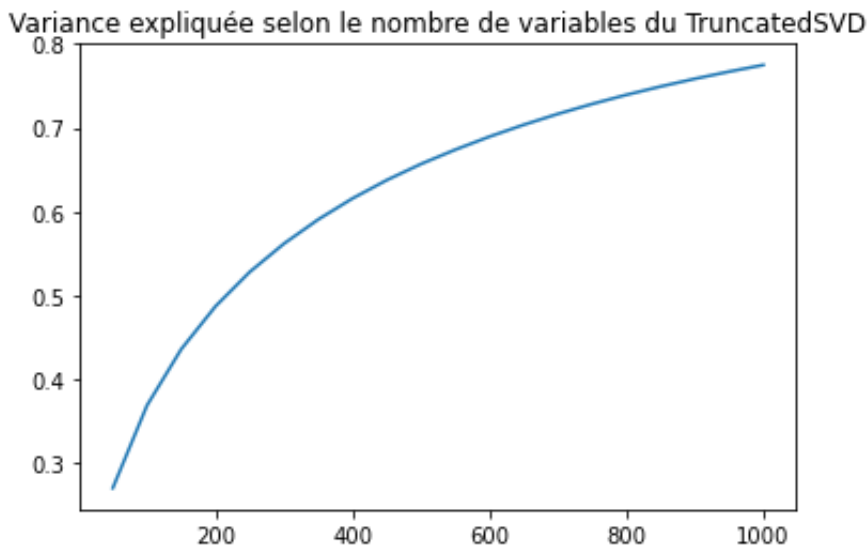
Pour les stopwords comme pour la racinisation, il faut bien sûr s'adapter à la langue du texte, en l'occurrence l'anglais. C'est donc ce que j'ai fait.

Ces transformations effectuées, il est maintenant temps de passer d'une liste de mots à des variables numériques, utilisables dans des analyses. J'ai pour cela utilisé CountVectorizer.

Cet objet transforme notre sac de mots en une liste d'occurrence de chaque mot. Le résultat est donc bien sûr composé principalement de 0, car un mot donné ne sera pas présent dans la majorité des textes individuels. L'objet obtenu est donc une matrice creuse contenant une très grande quantité de variables.

Vu la taille de la matrice des occurrences, il pourra être utile d'effectuer une réduction de dimension, pour diminuer le temps d'exécution des algorithmes. Il faudra dans ce cas déterminer le nombre idéal de dimensions, en examinant la variance expliquée pour chaque nombre de composantes.

Vu que nous avons une matrice creuse, j'ai utilisé TruncatedSVD plutôt qu'une analyse en composantes principales classique. Le graphique des variances expliquées obtenu est le suivant :



On voit que la variance expliquée augmente lentement par rapport au nombre de composantes. Il faudrait, pour avoir ne serait-ce que 80% de variance expliquée, utiliser plus de 1000 composantes, ce qui ne serait vraiment pas pratique.

Cependant, étant donné que nos variables représentent des mots, toutes ne sont pas importantes. Il est donc acceptable de n'utiliser que 50 composantes, pour obtenir une variance expliquée d'un peu plus de 25%.

Cela étant dit, s'il n'est pas nécessaire d'utiliser la réduction de dimension, il est bien sûr préférable d'utiliser toutes les données.

## Analyse non supervisée

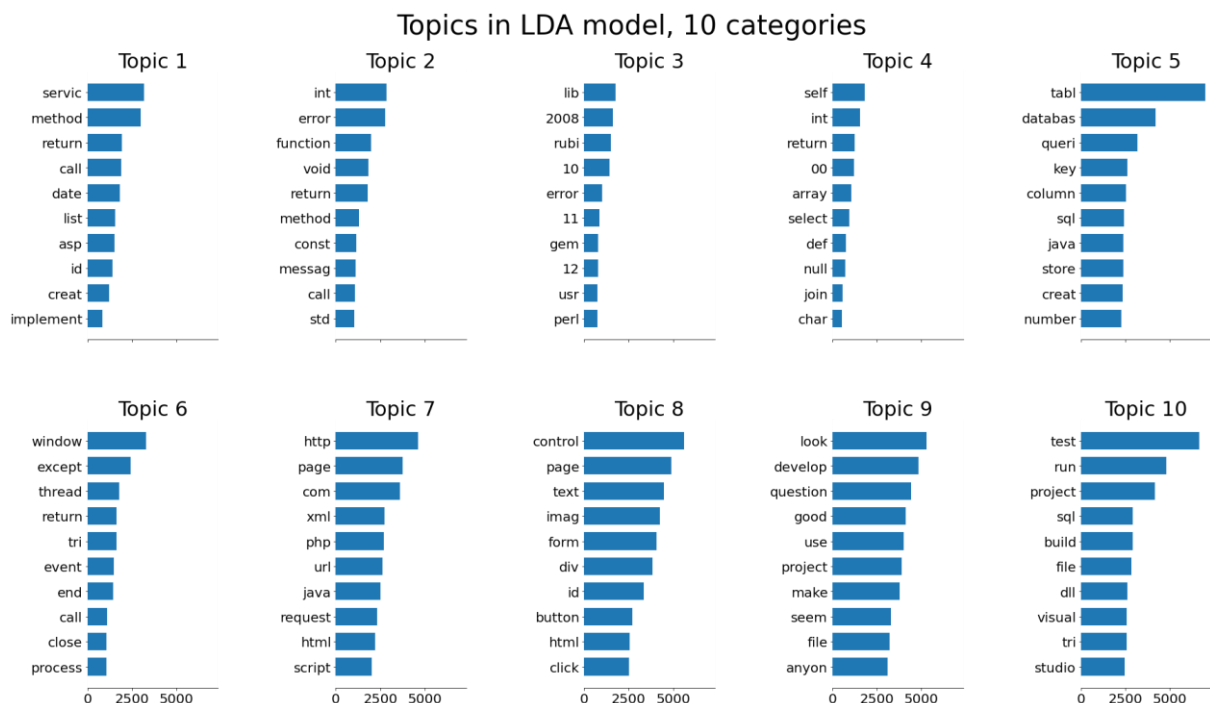
Nos données ayant été transformées en données numériques, nous pouvons maintenant passer à l'analyse non supervisée.

Le but de cette analyse est de regrouper les individus les plus similaires, de manière à former plusieurs groupes, qui seront chacun associés à une liste de tags. Nous utiliserons la Latent Dirichlet Association (LDA) et la Non-negative Matrix Factorisation (NMF).

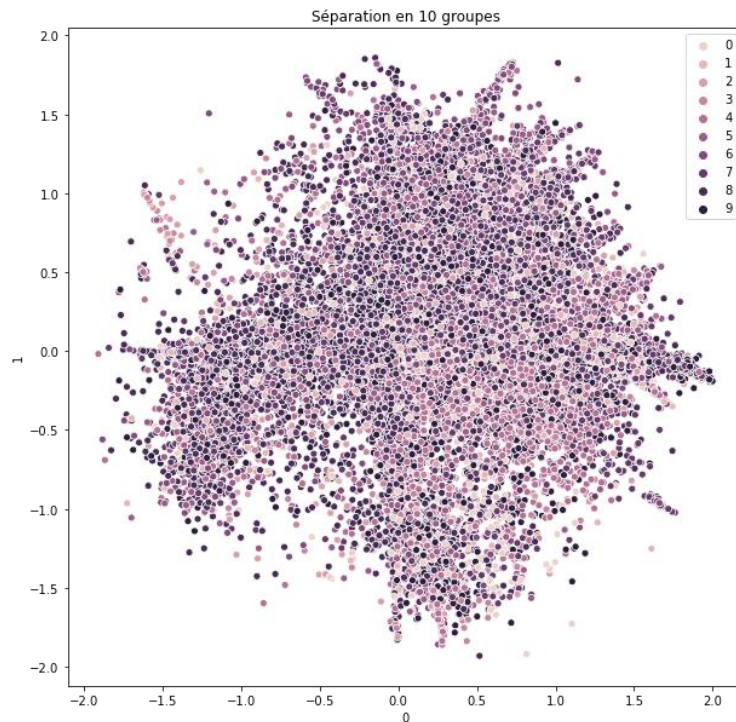
### LDA

La LDA suppose que chaque mot a certaine une probabilité d'appartenir à un groupe, et que chaque groupe est donc composé de probabilités sur chacun des mots. Le nombre de groupes est fixé à l'avance.

En testant avec plusieurs nombres de groupes, le résultat n'est pas très bon. Si l'on peut voir certains thèmes sur certains groupes, comme on peut le voir ci-dessous, le score de silhouette, qui mesure la proximité des membres au sein d'un groupe, est négatif quel que soit le nombre de groupes choisi au départ. Cela signifie que les individus sont moins proches des membres de leur groupes que des autres individus.



De plus, en examinant visuellement les individus, on ne voit aucun groupe apparaître sur le graphique.



Examiner les tags associés aux documents classifiés dans chacun des groupes ne fait apparaître aucune tendance, ce qui indique que cette classification n'a aucun lien avec les tags choisis par les utilisateurs.

Cela signifie que la LDA n'est pas adaptée pour ces données.

## NMF

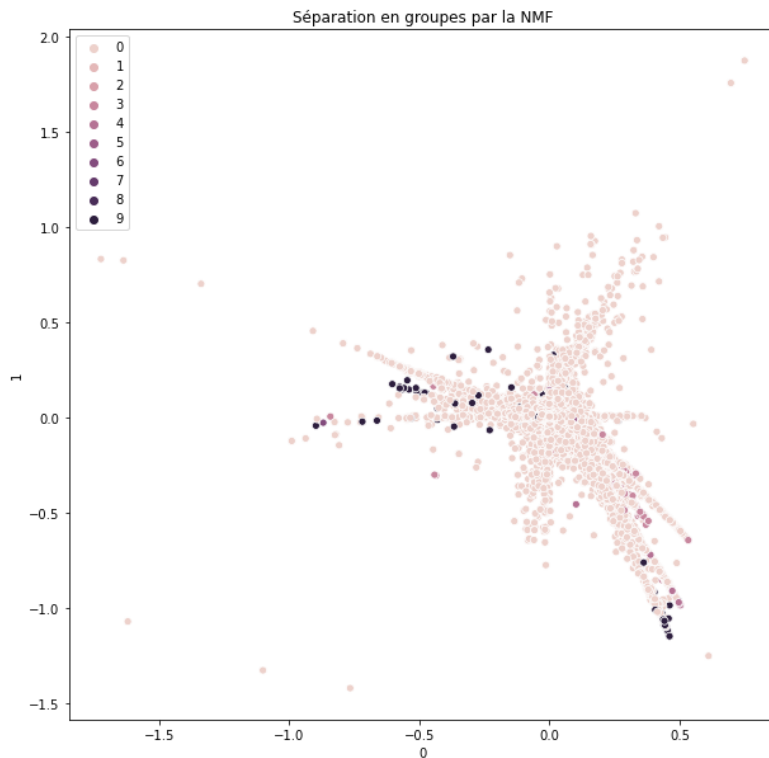
La NMF consiste à créer deux matrices qui permettront simultanément d'identifier les groupes et de classer les documents. Le nombre de groupe doit ici aussi être spécifié à l'avance.

Etant donné que la NMF nécessite une grande taille de mémoire, il est nécessaire d'utiliser une réduction de dimension. Comme indiqué plus haut dans ce rapport, cette réduction de dimension est une TruncatedSVD de 50 composantes.

En examinant les résultats de la NMF avec 10 classes, on constate d'abord que le score de silhouette est positif. Cependant, la représentation en deux dimensions (page 8) ne montre pas beaucoup plus les classes que dans le cas de la LDA.

Enfin, les tags associés aux classes obtenues montrent que les bons résultats du score de silhouette sont dus au fait que la quasi-totalité des individus sont classés dans le même groupe.

Cette classification n'est donc pas non plus efficace.



## Analyse supervisée

### Multiclass ou multilabel

Pour une analyse supervisée, nous avons besoin d'une variable cible qui nous permettra d'attribuer des tags aux textes selon leur classe. Le choix le plus simple est de prendre les tags les plus fréquemment utilisés, et de créer une classe par tag. Nous utiliserons ici 20 tags pour ne pas prendre trop de temps d'exécution, mais il serait certainement possible d'en avoir beaucoup plus. Nous utiliserons, pour réduire le temps d'exécution, les données de dimension réduite.

Le problème posé par ce choix de classes est bien sûr qu'une question a généralement plusieurs tags qui lui sont associés. Il serait possible de réunir en une seule classe les tags fréquemment utilisés, cependant, cela n'éliminerait pas complètement le modèle.

Une meilleure solution est de faire en sorte qu'un document puisse être classifié dans plusieurs classes simultanément, c'est-à-dire du multilabel plutôt que du multiclass. Les modèles de classification binaires seront entraînés sur chacune des classes indépendamment des autres, nous donnant au final 20 modèles, soit un par tag.

Pour préparer les données aux algorithmes, il faudra donc commencer par créer une table composée de 0 et de 1, indiquant pour chaque couple individu-tag si le tag s'applique au texte de l'individu. Nous devrons également séparer les données en données d'apprentissage et données de test, une étape essentielle pour pouvoir estimer les erreurs de manière fiable.



## SVC

Le support vector classifier est une méthode de classification qui consiste à déterminer l'emplacement de la frontière entre deux classes, en se fiant uniquement aux points proches de la frontière. Cette méthode donne souvent de bons résultats.

Malheureusement, l'entraînement sur un tag prend plus de 20 minutes. Etant donné que cet entraînement devrait être effectué pour chacun des 20 tags, et que nous voudrions également tester différents hyperparamètres, cette méthode n'est pas utilisable de manière pratique dans le cas présent.

## Forêts aléatoires

L'algorithme des forêts aléatoires entraîne un grand nombre d'arbres de décision, et prédit le score attribué à un individu pour une classe donnée en fonction du pourcentage d'arbres qui le mettraient dans cette classe.

L'arbre de décision possède un grand nombre d'hyperparamètres, mais pour gagner du temps, nous concentreront notre recherche sur deux seulement : le critère sur lequel se baseront les arbres pour créer les nœuds, et le nombre de variables considérées à chaque nœud.

Pour trouver quels hyperparamètres sont les meilleurs, il faut choisir un test permettant de comparer les modèles. J'ai choisi l'exactitude, ou accuracy, qui indique le pourcentage de bien classés, et le score f1, qui indique à quel point les nombres de faux négatifs et de faux positifs sont similaires.

Une difficulté se pose cependant : ces scores sont calculés sur un modèle, mais nous en avons ici entraîné 20 pour chaque couple d'hyperparamètres. Comment, donc, obtenir un score pour l'ensemble des 20 modèles ?

La solution que j'ai choisie consiste à faire la moyenne des scores sur chaque modèle. Cela nous permettra de trouver un modèle à peu près aussi bon sur chaque tag.

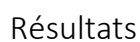
D'après les scores, les meilleurs hyperparamètres sont le critère entropy, et le nombre de variables sqrt.

Un autre hyperparamètre a de l'importance dans les forêts aléatoires : le nombre d'arbres. Ce paramètre améliore généralement le modèle quand il augmente, c'est pourquoi nous n'avons pas cherché à le fixer en même temps que les autres.

En examinant les modèles selon le nombre d'arbres, on s'aperçoit que les résultats ne s'améliorent pas. Nous n'avons donc pas besoin d'utiliser plus d'arbres.

La plupart des individus n'appartiennent pas à la classe d'un tag donné. Cela nous pose un problème, car cela signifie que le modèle aura tendance à ne classer la plupart des individus dans aucune classe. Le système de suggestion de tags ne proposerait alors aucun tag à l'utilisateur.

F-score selon le seuil choisi, où le rappel est 2 fois plus important que la précision



Un examen des matrices de confusion pour chacun des 20 modèles nous montre que les résultats varient selon le tag, certains contenant d'avantage de faux positifs, et d'autres de faux négatifs.

La visualisation des mots les plus fréquents parmi les individus appartenant à une des classes est sensiblement différente de celle montrant tous les mots du corpus.

## API

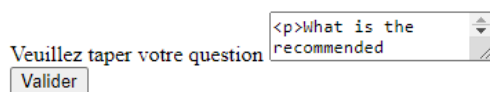
Pour implémenter le système de suggestion, il est nécessaire de créer une API.

L'interface homme-machine est une page html, dans laquelle il pourra taper la question pour laquelle il souhaite des suggestions de tags.

En arrière-plan, des fonctions transformeront d'abord le texte en utilisant toutes les transformations vues dans la partie traitement de données : stopwords, racinisation, réduction de dimension... Le texte sera ainsi dans le même format que les données qui ont servi à entraîner le modèle.

Se fera ensuite la prédiction de tags, prédiction qui sera alors transformée en une phrase informant l'utilisateur des tags suggérés. Cette phrase s'affiche enfin dans l'interface, comme on peut le voir ci-dessous :

### **Ce programme vous aidera à créer des tags pour vos questions.**



Veillez taper votre question

Tags suggérés : c#, .net, visual-studio.

Cette API est lancée en utilisant le code suivant :

```
FLASK_APP=MAIN.PY FLASK RUN --HOST=0.0.0.0 --PORT=8000
```

L'IHM est ensuite accessible à cette adresse :

[HTTP://EC2-18-222-168-221.US-EAST-2.COMPUTE.AMAZONAWS.COM:8000/](http://EC2-18-222-168-221.US-EAST-2.COMPUTE.AMAZONAWS.COM:8000/)

## Conclusion

Bien qu'ayant des imperfections, le modèle choisi permet de prédire de manière assez fiable les tags qu'ont choisi les utilisateurs, et donc de faire des suggestions pertinentes.