Catégorisez automatiquement des questions

Azim Makboulhoussen

OpenClassrooms
Projet 6 – Parcours Data Scientist
23 Mai 2018

Résumé.

Ce document décrit les approches utilisées afin d'implémenter un système de recommandation de tags pour le site Stack Overflow. Des modèles d'apprentissage supervisé et non supervisé ont été testés dans le cadre de ce projet de traitement automatique de langage naturel.

Table des matières

I. Introduction

Stack Overflow est un site web de questions et réponses sur des thèmes de la programmation informatique. Il est particulièrement réputé et utilisé par les développeurs. Il leur permet de poser des questions techniques et de recevoir des réponses et de l'aide des membres de la communauté.

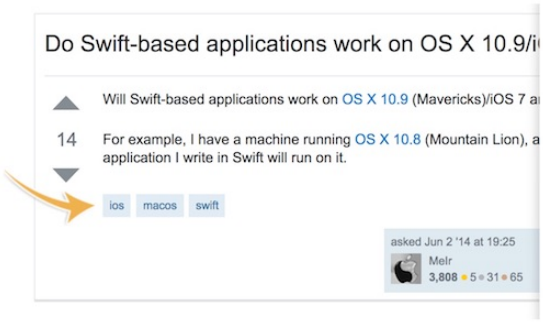


Figure 1 : Exemple de tags

Stack Overflow utilise un système de mot clés (tags) qui permet le classement des questions mais qui facilite aussi les recherches. Chaque question est associée à un ensemble de tags en lien avec les sujets traités.

Afin d'aider les utilisateurs novices du site, nous nous proposons de développer un système capable de recommander des tags pour une nouvelle question.

Pour cela, nous nous appuierons sur les techniques de l'apprentissage machine. Nous commencerons par manipuler des données textuelles afin de les rendre exploitables par les algorithmes, puis nous évaluerons différentes modélisations (supervisées et non supervisées) et implémenterons celle donnant la meilleure performance.

Ce document décrit la démarche que nous avons suivie pour répondre à cet objectif ainsi que la solution que nous avons implémentée.

II. Les données

1. Récupération des questions

Notre modélisation sera basée sur les données des questions. Il faut donc dans un premier temps les récupérer depuis le site de Stack Overflow. Celui-ci propose une interface d'export des données basée sur des requêtes SQL.

En regard de la volumétrie des données, une limite du temps d'exécution est fixée par l'outil. Pour pouvoir récupérer un nombre de questions suffisant pour notre modélisation, nous avons fait plusieurs requêtes avec les contraintes suivantes :

- Filtre sur les questions ayant un score (une notation) supérieure à 5 pour avoir des questions pertinentes.

- Filtre sur les questions avec POSTTYPEID=1 pour n'avoir que les objets de type questions.
- Utilisation du champ ID pour avoir des sous ensemble de données.

Exemple de requête :

```
SELECT TITLE, BODY, SCORE, TAGS
FROM POSTS
WHERE POSTTYPEID=1
AND ID >= VAL_1 AND ID < VAL_2
AND SCORE > 5
```

2. Caractéristiques des données

Notre ***dataset*** est composé d'un peu plus de 64 000 questions (POSTS).

	TITLE	BODY	SCORE	TAGS
0	Java generics variable <T> value	<p>At the moment I am using the following code...	6	<java><generics>
1	How a value typed variable is copied when it i...	<blockquote>\n <p>Swift's string type is a va...	6	<swift><function><value-type>
2	Error while waiting for device: The emulator p...	<p>I am a freshman for the development of the ...	6	<android><android-studio><android-emulator><avd>
3	gulp-inject not working with gulp-watch	<p>I am using gulp-inject to auto add SASS imp...	10	<javascript><node.js><npm><gulp><gulp-watch>
4	React - Call function on props change	<p>My TranslationDetail component is passed an...	12	<reactjs><react-router>

Figure 2 : les 5 premières lignes de nos données

Pour chaque question, nous avons le **titre**, le **contenu**, le **score** et les **tags** associés à celle-ci.

Variable	Type	Description
TITLE	Textuelle	Titre du post (de la question)
BODY	Textuelle	Contenu textuel du post (détail de la question)
SCORE	Continue (int)	Score données par les internautes pour la pertinence de la question
TAGS	Textuelle	Les tags associés au post. Liste de tags (chaque tag est entre '<' '>'

Notre ***dataset*** ne comporte aucune donnée vide ce qui est une bonne chose ☺.

3. Prétraitement des données textuelle

Pour pouvoir exploiter efficacement nos données, nous devons préalablement faire un certain nombre de traitements. Cette partie décrit les opérations que nous avons effectués sur les données textuelles (TITLE et BODY).

a) Suppression des tags HTML

La données de la colonne BODY sont stockées au format HTML et contiennent des tags HTML inutile qui n'apporte aucune valeur dans l'analyse de texte. Nous

supprimons donc dans un premier temps tous ces tags pour ne conserver que le contenu. Nous nous basons pour cela sur la librairie ***BeautifulSoup*** de Python qui facilite grandement ce type de traitement.

b) Traitement du texte

- Suppression des accents et normalisation des données encodées en Unicode au format ASCII :

```
text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
```

- Nous avons ensuite remis les extensions (langue Anglaise) sous une forme classique (exemple : Don't ➔ Do not). Pour cela nous nous sommes basés sur du code récupéré depuis ce lien

- Nous avons ensuite
  - passé tout le texte en minuscule,
  - supprimé tous les caractères non alphabétiques
  - et retiré tous les mots à un seul caractère. Ils n'apporteront rien dans l'analyse.

  Pour ces différents traitements, nous avons utilisés des opérations à base d'expressions régulières.

c) Tokenization

Nous allons ensuite segmenter le texte en unité « atomique » appelé tokens. Le package NLTK sera utilisé pour tokeniser les documents de notre corpus en mots.

```
tokens = tokenizer.tokenize(text)
tokens = [token.strip() for token in tokens]
```

d) Suppression des mots vides (stopwords)

Certains mots ne nous aideront pas réellement à déterminer les tags associées. Des mots comme ***the***, ***a*** et ***and*** ne permettent pas de distinguer les questions. En les supprimant, nous allons également réduire la taille de nos données et par extension améliorer les performances de nos algorithmes d'apprentissage. Ces mots sont appelés **stopwords.**

Nous utilisons la libraire **NLTK** pour supprimer les ***stopwords*** de notre texte.

```
meaningful_words = [token for token in tokens if token not in stopword_list]
```

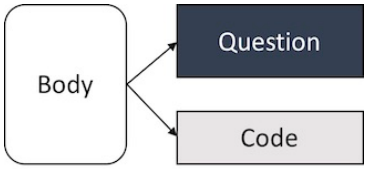
e) Racinisation (Stemming)

Le stemming (racinisation en français) vise à garder la racine des mots étudiés. L'idée étant de supprimer les suffixes, préfixes et autres des mots afin de ne conserver que leur origine. C'est un procédé fréquent dans les applications de traitement automatique du langage naturel.

Dans notre cas, nous avons utilisé le stemming basé sur l'algorithme de Porter :

```
porter = PorterStemmer()
stemmed = [porter.stem(word) for word in meaningful_words]
```

4. Traitement particulier du contenu de type code



La colonne BODY contient quelques fois du code entouré des balises <code>. Nous ne pouvons appliquer les traitements précédents car nous risquons de perdre certaines données utiles pour la prédiction de tags. Nous avons donc effectuer un traitement plus simple sur ces données :

- Suppression des tags HTML
- Suppression des signes de ponctuation
- Seuls les caractères alphanumérique sont conservés
- Séparation en mots qui sont passés en minuscule

5. Traitement de la colonne TAGS

Les tags sont séparés par les caractères '<' et '>'.

Exemple de tags :
<swift><function><value-type>

Toujours en nous appuyons sur les expressions régulières, nous avons séparé chacun des tags et nous les avons stockés dans une structure de type liste.

```
def getTagNames(text) :
    tags = ' '.join(re.findall('<(.*?)>',text))
    return tags.split(' ')

df['TAGS_P']=df['TAGS'].apply(lambda x: getTagNames(x))
```

Exemple de transformation :

```
<android><android-studio><android-emulator> ➔ [android, android-studio, android-emulator]
<reactjs><react-router>                     ➔ [reactjs, react-router]
```

6. Analyse des données

Nous avons complété notre exploration des données par une analyse de chacune des *features* de nos données. Nous n'allons pas détailler cette analyse dans ce document mais simplement nous concentrer sur notre variable cible : les tags. Nous vous invitons à consulter le notebook d'exploration pour l'étude complète.

Les tags

Les questions ont en moyenne 3 tags et la moitié de nos jeux de données possède entre 2 et 4 tags comme nous pouvons le voir dans le box plot suivant :

- Une question possède au minimum 1 tag et 5 tags au maximum.

Certains tags tel que ***javascript, androïd, python et java*** sont particulièrement présents comme l'illustre l'histogramme ci-dessous :

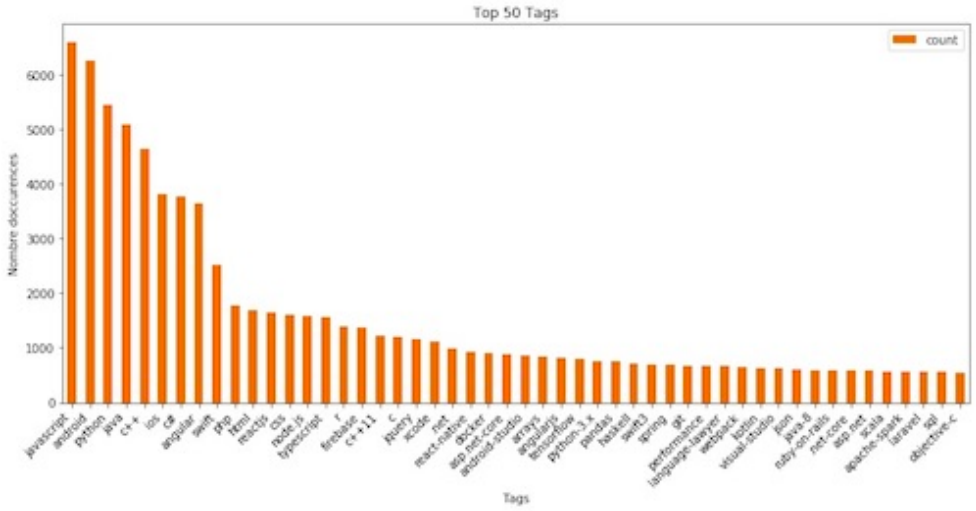


Figure 3 : Occurrences des 50 tags les plus fréquents

Nous constatons également que dans près de **73%** des cas, nous retrouvons au moins un tag associé à la question soit dans le ***titre*** ou le ***body*** de celle-ci.

III. Notre démarche

1. Techniques de modélisation

Pour pouvoir suggérer des tags relatifs au contenu d'une question, nous avons essayé différentes techniques d'apprentissage machine.

- La première basée sur l'apprentissage non supervisée pour classifier nos documents.

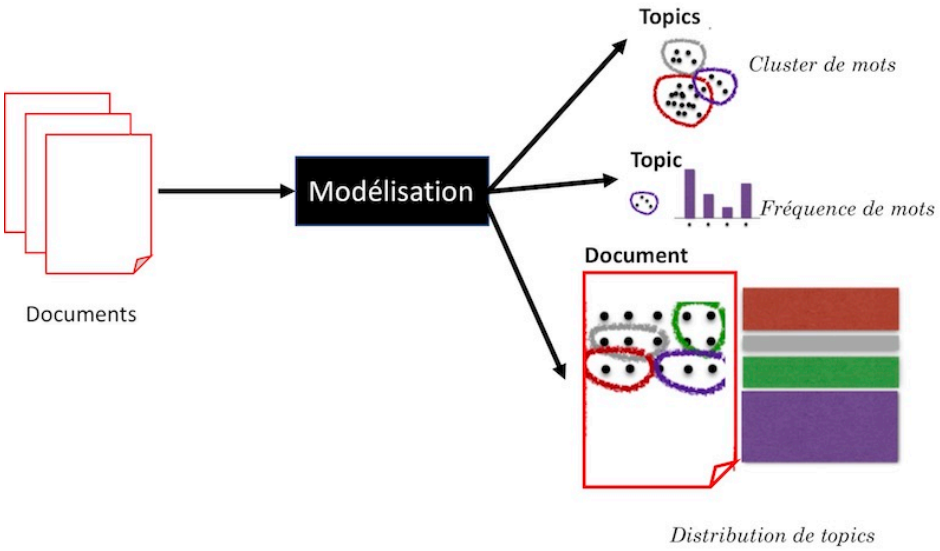


Figure 4 : Topic modeling

- Cette technique est appelé « Topic Modeling » dont le principe est d'identifier automatiquement des sujets latents (topics) abordés dans un corpus de documents et assigner les sujets détectés à ces différents documents.
- Un « topic » correspond à un ensemble de mots qui apparaissent fréquemment dans notre corpus.
- Nous déterminerons alors à partir de ces topics affectés aux documents les tags les plus pertinents.

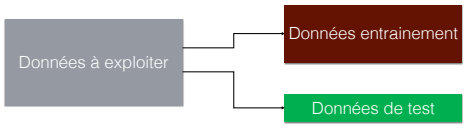
➢ La deuxième basée sur l'apprentissage supervisée classique. A partir de *features* extraites des documents, nous allons entrainer nos « *classifiers* » à apprendre à prédire nos tags.

2. Filtre sur les tags fréquents

Notre corpus est composé de plus de **14 000** tags différents. Afin d'améliorer les performances d'exécution et de proposer des tags cohérents, nous avons fait le choix de ne conserver que les tags qui sont présents dans au moins 10 documents. Cette liste de tags fréquents comprend un peu plus de **2 000** tags.

Nos données d'entrainements seront donc filtrés sur cette liste.

3. Séparation des données



Nous allons séparer nos données en jeu d'apprentissage et de tests. Le premier jeu de données sera utilisé pour l'entrainement de nos algorithmes et le deuxième pour

comparer les modèles et vérifier la qualité des prédictions. On utilisera ainsi des données

Les données de tests ne seront pas filtrés sur les tags les plus fréquents et permettront donc de tester nos modèles avec des questions issues de cas réels.

IV. Transformation des données textuelles en matrice documents-mots

Notre objectif est d'appliquer un modèle d'apprentissage capable de prédire des tags à partir du contenu des question. Cependant les algorithmes d'apprentissage automatique ne fonctionnent pas avec du texte brute mais plutôt avec des nombres. Il nous faut donc traduire nos questions en quelque chose de compréhensible par un algorithme. C'est ce que nous allons faire dans cette partie.

1. La représentation Bag-of-Words

Le « Bag of Words » est une représentation de document dans laquelle on indique les occurrences des mots le composant. Elle se base sur un dictionnaire des mots du corpus et pour un document donné, chaque mot se voit affecté une mesure de sa présence dans le document.

Un document est donc représenté par un vecteur de la taille du vocabulaire dont la composante *i* indique la fréquence du *ième* mot de notre vocabulaire dans le document. Une représentation bag-of-words utilisera la matrice composée de l'ensemble des N documents qui forment le corpus comme entrée de nos algorithmes.

Exemple :

Si j'ai 3 documents :

- D1 : projet data-science
- D2 : data-science : soutenance data-science
- D3 : soutenance projet bientôt

On aura la matrice suivante :

	Projet	Data-science	Soutenance	Bientôt
D1	1	1	0	0
D2	0	2	1	0
D3	1	0	1	1

Nous utiliserons cette représentation notamment avant d'appliquer la modélisation LDA (latent Dirichlet allocation) que nous verrons plus loin dans ce document.

2. La représentation en n-grams

Un N-grams est simplement une séquence de N-tokens de mots. Un 2-gram ou bi-gramme est une séquence de deux mots comme "java language", "python module" et

un 3-gram ou trigramme est une séquence de 3 mots. Le principe reste le même que le « Bag-of-Words » mais chaque composant de notre vecteur correspond à une séquence de mots. Le « Bag-of-Words » est en réalité un 1-gram. Dans notre projet, nous avons utilisé la représentation en uni-gramme et bi-gramme notamment avant d'appliquer les algorithmes d'apprentissage supervisée.

3. La représentation en TF-IDF

Nous l'avons vu dans le « Bag-of-Words », une fois le vocabulaire défini, il faut compter les occurrences des mots dans les documents. Le problème avec cette technique c'est que les mots les plus présents auront un score plus importants alors qu'ils n'apporteront pas forcément autant d'information à nos modèles d'apprentissage que d'autres mots plus rares mais d'avantage liés au domaine couvert par le document.

Une technique pour contourner ce problème est de pondérer l'occurrence d'un mot dans un document par sa fréquence d'apparition dans tous les autres documents. Les mots qui sont très présents (exemple : « un », « le ») seront alors d'avantage pénalisés. Cette approche est appelé : Term Frequency – Inverse Document Frequency (TF-IDF) :

- Term Frequency : fréquence du terme dans le document analysé
- Inverse Document Frequency : fréquence inverse de document (mesure importance du mot dans l'ensemble du corpus)

La formule mathématique de calcul du score TF-IDF est la suivante :

$$w_i = tf_{ij} \times \log(\frac{N}{df_i})$$

tf_{ij} : Nombre de fois que mot i apparaît dans document j / nombre mots dans j
N : nombre total de documents
df_i : nombre de documents contenant i

Nous avons utilisé cette représentation pour l'apprentissage supervisé mais également non supervisé avec l'algorithme NMF (factorisation par matrices non négatives).

V. Modélisation non supervisée

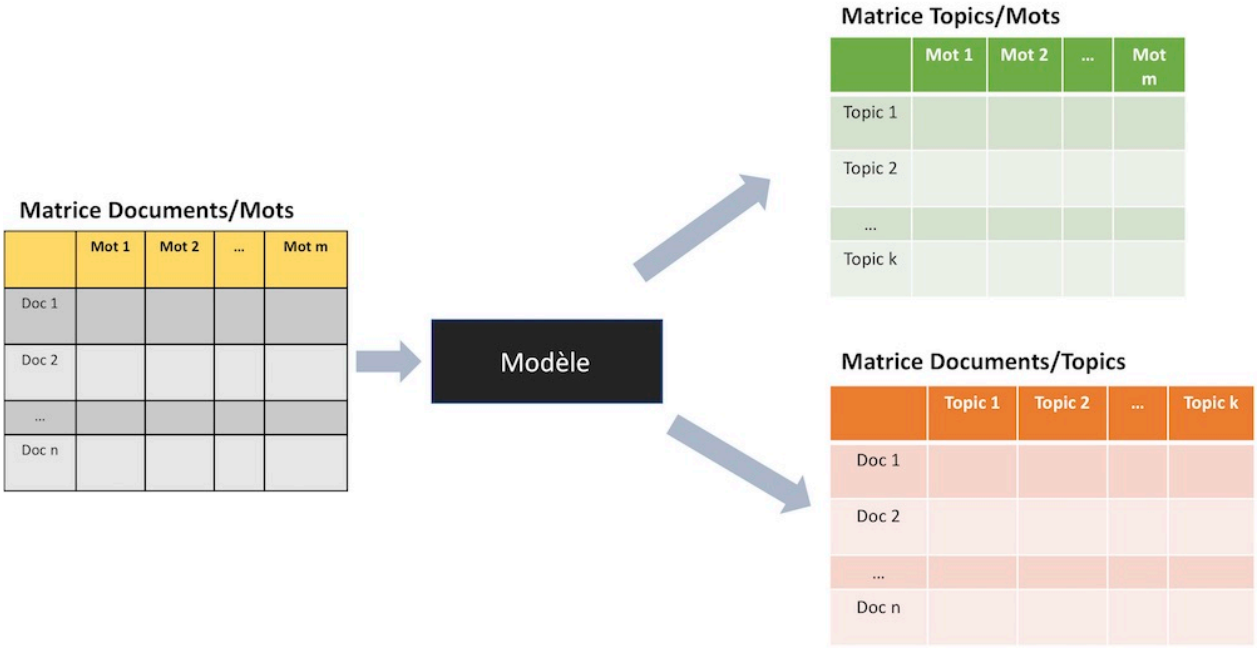
1. Le principe utilisé

Comme expliqué dans la section III, nous avons utilisé les techniques de Topic Modeling qui est une approche d'apprentissage non supervisée pour découvrir des sujets (topics) qui sont présents dans notre corpus.

Apprentissage

Nous avons testé 2 algorithmes de « Topics Modeling » : le LDA et le NMF. Les 2 algorithmes prennent en entrée une matrice Documents / Words (Bag of Words) et génère en sortie 2 matrices :

- Une matrice associant les documents aux topics
- Une matrice associant les topics aux mots de notre vocabulaire



Ces algorithmes ne savent pas déterminer le nombre de topic optimal de manière automatique. Il nous faut donc spécifier le nombre de topics souhaités.

Prédiction

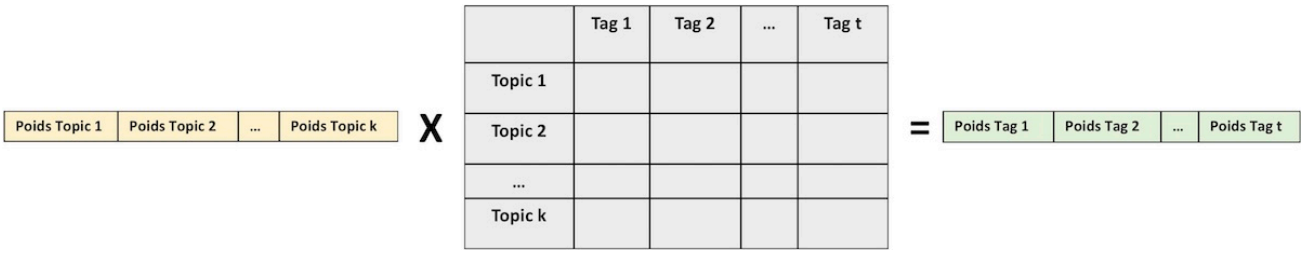
Afin de prédire les tags, nous avons construit une nouvelle matrice qui associe les topics aux tags.

Pour cela, pour chaque **tag i** et chaque **topic j,** nous avons sommer la probabilité d'appartenance au **topic j** des documents contenant le **tag i**. Nous obtenons alors la matrice suivante :

	Tag 1	Tag 2	...	Tag t
Topic 1				
Topic 2				
...				
Topic k				

Figure 5 : Matrice Topics - Tags

Nous appliquons ensuite notre modèle sur une question afin qu'il détermine la prédominance des topics dans celle-ci. Nous obtiendrons en résultat un vecteur de poids donné à chacun des topics.
En multipliant ce vecteur à notre matrice Topics/Tags, nous pourrons alors déterminer les n tags les plus pertinents en prenant les n tags ayant les meilleurs scores.



Évaluation

Pour évaluer la qualité de nos algorithmes, nous avons prédit pour chacune des questions de notre jeu de tests les 5 tags ayant les meilleurs scores. Nous avons calculé un score en faisant la moyenne du pourcentage de tags correctement prédit pour chaque question.

$$score_i = \frac{T_i}{N_i}$$

$score_i$ = score de prédiction de la question i
T : nombre de tags identiques aux tags réels de la question i
N : nombre de tags total réel de la question i

2. LDA

Le **Latent Dirichlet Allocation** (LDA) est une algorithme très répandu de « Topic Modeling ». LDA est un modèle probabiliste et qui fonctionne de manière itératif comme décrit ci-dessous :

- Chaque mot de chaque document est affecté à un des K topics de manière aléatoire
- Pour chaque document d :
  - Et pour chaque mot w dans d :
    - Et pour chaque topic t, calculer la probabilité p1(topic t | document d) = proportion de mots dans le document d qui sont affecté au topic t et la probabilité p2(mot w | topic t) : la proportion de topic t attribué au mot w sur l'ensemble du corpus.
    - On affecte un nouveau topic à w avec une nouvelle probabilité qui est le produit de p1 et p2
- L'étape précédente est effectuée de manière itérative et permet ainsi d'améliorer la qualité des affectations.

- Nous avons utilisé l'objet *CountVectorizer* pour transformer nos données en matrice documents / mots.
- Nous nous sommes appuyés sur la recherche sur grille pour optimiser notre algorithme notamment en essayant différentes valeurs pour les hyper-paramètres suivants :
  - ***min_df*** : nombre minimum de documents dans lequel le mot doit être présent pour être retenu dans notre vocabulaire
  - ***max_df*** : nombre maximum de documents dans lequel le mot doit être présent pour être conservé dans le vocabulaire
  - ***n_components*** : nombre de topics souhaité

- Nous avons également faire varier le nombre de tags à prédire pour comparer les résultats.

Résultat :

Nous obtenons avec cet algorithme un score de prédiction de **24,65%** avec nos données de tests. En moyenne 24% des tags sont donc correctement prédit.

3. NMF

L'algorithme **NMF** (Non Negative Matrix Factorization) est un modèle algébrique linéaire qui factorise les vecteurs à hautes dimensions en une représentation plus réduite.

- Nous avons utilisé la réprésentation TF-IDF pour transformer nos données en matrice documents / mots.
- Nous nous sommes appuyés sur un tuning manuel des hyper-paramètres afin d'optimiser notre algorithme.

Résultat :

Nous obtenons avec cet algorithme un score de prédiction de **26,83%** avec nos données de tests. Ce score est meilleur que celui avec obtenu avec le LDA.

VI. Modélisation supervisée

1. Le principe utilisé

Nous avons ensuite testé des algorithmes d'apprentissage supervisé pour suggérer les tags. Notre variable cible est donc la « feature » ***TAGS***. C'est une variable catégorielle, nous devons donc utiliser les algorithmes de classification supervisé.

Variables d'entrée

Comme pour les algorithmes non supervisés, nous utiliserons les données TITLE et BODY représenté sous forme matricielle (TF-IDF) pour l'apprentissage de nos algorithmes.

Cible multi labels

Nous avons une particularité dans notre cas car pour chaque question, nous ne devons pas prédire un tag mais un ensemble de tags. Nous sommes dans un cas de problème de classification multi-label.

Fort heureusement, la librairie ***sklearn*** implémente la résolution de ce type de problème. Nous avons utilisé la classe MultiLabelBinarizer qui transforme notre variable cible en une matrice binaire indiquant la présence d'un tag (chaque colonne représente un tag).

[0, 1, 1] ➔ présence tag 1 et tag 2
[1, 0, 0] ➔ présence tag 0
[1, 1, 0] ➔ présence tag 0 et tag 1

One-Vs-The-Rest

La librairie sklearn implémente également la stratégie « one-vs-all » (seul contre tous) au niveau de sa classe OneVsRestClassifier et gère l'apprentissage multi-labels à partir d'une matrice dont la cellule [i, j] prend la valeur 1 si le label j est présent dans l'échantillon i.

Évaluation

Pour évaluer la qualité de nos algorithmes, nous avons utilisé la même solution que pour les algorithmes non supervisés.

2. Les algorithmes

- Les algorithmes suivants ont été testés sur notre jeu de données :
  - SVM Linéaire optimisé avec une descente de gradient stochastique.
  - Gaussian Naive Bayes
  - Decision Tree
  - Random Forest
  - Gradient Boosting

- La recherche sur grille et la validation croisée ont été utilisées pour améliorer les performances de nos algorithmes et trouver les meilleures valeurs des hyper-paramètres.

VII. Résultats

Modèles supervisés

	Gaussian Naive Bayes	Decision Tree	SGD	Random Forest	Gradient Boosting
Scores	15.64 %	50.43 %	**55.01 %**	40.06 %	48.07 %

Modèles non supervisés

	LDA	NMF
Scores	24.65 %	26.83 %

Nous obtenons le meilleur score et une meilleure performance avec un SVM optimisé par un SGD.

Nous avons implémenté à partir de ce modèle, une interface WEB . Celle-ci va proposer à l'utilisateur 7 tags relatifs au contenu de sa question.

On constate que plus on augmente le nombre de tags à prédire, plus le score de prédiction est meilleur. On arrive à un score de 57% avec 8 tags. Cependant nous pensons qu'au-delà de 7, le nombre de tags suggéré à l'utilisateur risque d'apporter plus de confusion que de l'aide. Nous nous sommes donc limité à 7 tags dans notre interface.

L'interface WEB est accessible depuis : https://suggesttags.herokuapp.com/

VIII. Conclusion

L'objectif de ce projet était d'implémenter un système de suggestion de tags pour le site Stack Overflow. Nous avons pu essayer différentes approches de traitement automatique de langage naturel et d'apprentissage machine.

Nous avons sélectionné un algorithme basé sur l'apprentissage supervisée, le machine à vecteurs de support (SVM) optimisé avec une descente de gradient stochastique (SGD) qui a donné les meilleurs résultats dans nos tests.

D'autres pistes devront être essayées et qui pourraient apporter des améliorations dans les prédictions.

- Par exemple l'utilisation des prolongements de mots récurrents (word embeddings) et les réseaux de neurones.
- Nous pourrons également exploiter l'historique de l'utilisateur pour lui proposer des tags qui sont d'avantage en lien avec les domaines de ces précédents questions.
- Enfin, il serait intéressant de pouvoir proposer de nouveaux tags qui émergent avant qu'ils soient utilisés.