

## Catégorisez automatiquement des questions

### PROJET 6 DATA SCIENTIST

**Résumé :** Ce rapport présente les différentes approches que j'ai mises en œuvre afin de proposer une API de catégorisation des questions. Voici les différents éléments qui seront détaillés :

- Requêtage des questions sur le site Stackoverflow
- Traitement de document texte avec nettoyage et création des mots utilisés (CORPUS)
- Modélisation afin d'obtenir les propositions de catégorie
- Enfin déploiement de l'API

WALROFF Jérôme | [jerome.walroff@gmail.com](mailto:jerome.walroff@gmail.com) | 10 janvier 2022

## Table des matières

Catégorisez automatiquement des questions.....	0
1. Requêtage .....	3
2. Préprocessing des POSTS.....	4
1) Nettoyage .....	4
2) Bag of words vs Tf-IDF .....	6
3) Modélisation.....	8
1) Approche non supervisée .....	8
2) Approche supervisée .....	11
4) Déploiement API .....	14
5) Git.....	14
6) Conclusion .....	14

# GLOSSAIRE

Terme	Description	Lien
<b>NLTK</b>	<b>Natural Language ToolKit.</b> Librairie pour traiter le langage naturel	<a href="https://www.nltk.org/">https://www.nltk.org/</a>
<b>Beautiful Soup</b>	<b>Beautiful Soup</b> est une bibliothèque Python permettant d'extraire des données de fichiers HTML et XML	<a href="https://beautiful-soup-4.readthedocs.io/">https://beautiful-soup-4.readthedocs.io/</a>
<b>Bag of Words (Sac de mots)</b>	<b>Sac de mots</b> : Fréquence du mot dans le document	<a href="https://fr.wikipedia.org/wiki/Sac_de_mots">https://fr.wikipedia.org/wiki/Sac_de_mots</a>
<b>Tf-IDF</b>	<b>Term frequency-inverse document frequency</b> : Fréquence du terme par rapport à la fréquence du terme dans l'ensemble du copus	<a href="https://fr.wikipedia.org/wiki/TF-IDF">https://fr.wikipedia.org/wiki/TF-IDF</a>
<b>NMF</b>	Non-negative matrix factorization	<a href="https://en.wikipedia.org/wiki/Non-negative_matrix_factorization">https://en.wikipedia.org/wiki/Non-negative_matrix_factorization</a>
<b>LDA</b>	Latent Dirichlet Allocation	<a href="https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24">https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24</a>
<b>SPACY</b>	Librairie PYTHON de traitement du langage naturel	<a href="https://spacy.io/">https://spacy.io/</a>
<b>Naive Bayse</b>	Type de classification bayésienne. Modèle à caractéristiques statistiquement indépendantes	<a href="https://fr.wikipedia.org/wiki/Classification_na%C3%AFve_bay%C3%A9sienne">https://fr.wikipedia.org/wiki/Classification_na%C3%AFve_bay%C3%A9sienne</a>

## 1. Requêtage

Le site Stackoverflow propose un outil de requêtage de type SQL afin de rechercher par différents critères des questions.

Je ne vais pas détailler ici le langage SQL qui n'est pas le propos de ce rapport. Je vais expliquer les critères fonctionnels que j'ai retenus pour sélectionner mon jeu de données.

Afin de constituer mon jeu de données, je dois sélectionner le titre et le corps de la question qui constituent mon entrant. La prédiction étant le Tags que je dois proposer à l'utilisateur. Par conséquent ces trois champs seront dans mon jeu de données.

Par la suite j'ai cherché à récupérer des questions de qualité en essayant le plus possible de ne pas prendre en compte les questions qui n'ont pas suscité d'intérêt au sein de la communauté. Pour cela j'ai utilisé le fait qu'une question avait généré une réponse. Même si Stackoverflow ne détaille pas le calcul du score, j'ai également utilisé cette information en ne prenant que les questions avec un score supérieur à 100.

Enfin je me suis assuré que tous les champs étaient non nuls cela pour faciliter mes traitements par la suite.

### **Piste non retenue :**

J'avais pensé retenir la description de l'auteur de la question en utilisant l'information contenue dans la table AboutMe proposée par Stackoverflow. J'ai finalement renoncé à son utilisation par faute de temps.

### **Requête définitive :**

```
SELECT P.Id,BODY,Title,Tags,P.CreationDate //Information souhaitée
```

```
FROM posts P //Table à requêter
```

```
where PostTypeId = 1 // Critères de sélection
```

```
AND AcceptedAnswerId is not null
```

```
AND score > 200
```

```
AND Tags is not null
```

**J'obtiens un jeu de données avec 16 413 questions que je vais étudier.**

## 2. Préprocessing des POSTS

Notre jeu de données ne possède pas d'éléments numériques. En effet ces éléments ont servi à sélectionner mes questions sans avoir d'intérêt pour la partie modélisation.

Comme vu précédemment je n'ai pas non plus d'éléments non renseignés car j'ai fait le nécessaire au niveau de ma requête pour ne pas le traiter par la suite.

L'enjeu du traitement de données dans notre contexte est le traitement du langage naturel. Ou comment à partir d'un texte nous allons créer des dimensions exploitables par des algorithmes de machine learning.

Pour ce faire j'ai exécuté les traitements suivants :

- Nettoyage et normalisation du texte
- Transformation
- Classification

### 1) NETTOYAGE

Les traitements présentés ci-dessous ont été appliqués à toutes nos dimensions Titre, Body et Tags. Quelques exceptions seront mises en évidences.

J'ai utilisé la librairie **NLTK (Natural Language ToolKit)**. Il s'agit d'une plateforme leader pour la création de langage Python à partir de données du langage humain.

Mise en minuscule : Afin de s'assurer que toutes les comparaisons réalisées par la suite se basent sur un texte similaire une mise en minuscule de l'ensemble des éléments a été réalisés.

Spécificité champs TAGS

- Expression régulière : Celui-ci est encapsulé dans une balise <...>. J'ai donc réalisé une expression régulière pour récupérer l'information contenue à l'intérieur et utiliser la fonction `word_tokenize` de NLTK
- L'idée au niveau des Tags était de ne retenir que les 100 plus utilisés. J'ai donc réalisé un comptage de la présence du tag pour constituer un Top100. J'ai alors supprimé tous les tags n'appartenant pas à ce Top100. Ceci a engendré des questions sans Tags que j'ai supprimées. Je me retrouve avec un jeu de données de 15 171 questions.

Champs TITRE et CORPS

- Dans ces champs l'utilisateur a pu écrire avec des balises html afin de mettre en évidences certains éléments du texte. Pour traiter ce point j'ai utilisé la librairie BeautifulSoup car NLTK ne propose de librairie sur le sujet.
- Suppression de la ponctuation
- Suppression des Stopwords : Au sein de chaque langage plusieurs mots permettent de lier les idées, d'introduire ou de conclure des phrases. Ces mots

n'apportent pas d'éléments de contexte sur le sujet traité. Par conséquent ils doivent être également supprimés. Ces mots sont appelés des Stopwords. Dans la librairie NLTK il est très simple de supprimer ces mots.

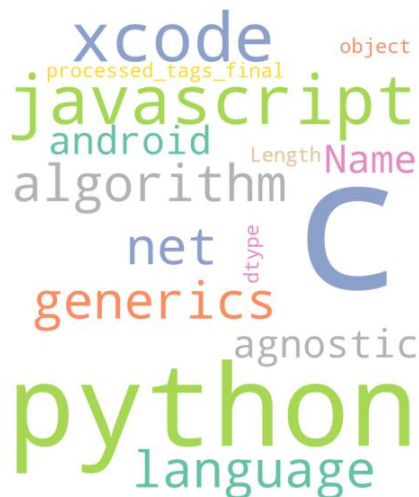
- Lemmatisation : Le dernier point consiste à trouver la racine des mots afin de ne conserver que cette information. Ceci permet de limiter grandement notre corpus final et de rapprocher bien plus de terme entre eux. Là encore la librairie NLTK nous est d'un grand secours en proposant des outils de lemmatisation.



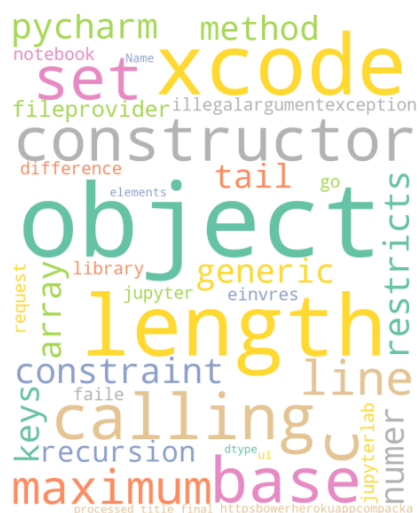
Je n'ai pas été convaincu par la lemmatisation. Dans un domaine avec beaucoup de termes techniques celle-ci n'apporte pas d'amélioration bien au contraire.

Le terme Jupyter devenait par exemple Jupyt

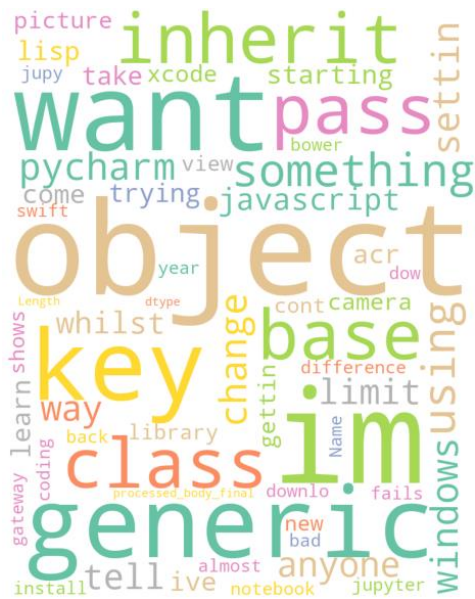
Voici la représentation des tags en nuage de mots



Voici la représentation des « Title » en nuage de mots



### Voici la représentation des « Body » en nuage de mots



## 2) BAG OF WORDS VS TF-IDF

Afin de traiter nos données dans les algorithmes de machine learning nous devons convertir le texte en une forme numérique. J'ai pu appréhender deux méthodes dans le cadre de ce projet : Bag of Words (Sac de mots) et Tf-IDF (term frequency-inverse document frequency).

Bag of Words est très simple en comptabilisant le nombre d'occurrence du mot dans le document. Si je prends un exemple sur 3 documents

Document 1 : La vie est belle

Document 2 : La vie est belle mais difficile

Document 3 : La vie est joyeuse

Documents	La	Vie	Est	Belle	Mais	Difficile	Joyeuse
Document 1	1	1	1	1	0	0	0
Document 2	1	1	1	1	1	1	0
Document 3	1	1	1	0	0	0	1

Par conséquent nous obtiendrons les trois vecteurs suivants :

Document 1 [1,1,1,1,0,0,0]

Document 2 [1,1,1,1,1,0]

Document 3 [1,1,1,0,0,0,1]

## Limitations

Volumes des vecteurs. Si le vocabulaire du document global est très important nos vecteurs seront de taille importante. De plus il génère des matrices creuses avec beaucoup de valeurs nulles.

N'apporte aucune information sur le sens du texte.

## TF-IDF

TFIDF fonctionne en augmentant proportionnellement le nombre de fois qu'un mot apparaît dans le document mais est contrebalancé par le nombre de documents dans lesquels il est présent. Cela pour minimiser la présence de mots fortement utilisés dans certain vocabulaire.

Nous avons d'un côté la fréquence, nombre de fois qu'un mot apparait dans le document divisé par le nombre total de mots.

De l'autre la fréquence inverse, nombre total de document divisé par le nombre de document avec le mot. Un logarithme est ajouté pour atténuer l'importance d'une valeur très élevée.

Si nous reprenons notre exemple nous avons

Documents	TF doc 1	TF doc 2	TF doc 3	IDF	TF-IDF Doc 1	TF-IDF Doc 2	TF-IDF Doc 3
La	$\frac{1}{4}$	$\frac{1}{6}$	$\frac{1}{4}$	$\text{Log} (3/3) = 0$	0	0	0
Vie	$\frac{1}{4}$	$\frac{1}{6}$	$\frac{1}{4}$	$\text{Log} (3/3) = 0$	0	0	0
Est	$\frac{1}{4}$	$\frac{1}{6}$	$\frac{1}{4}$	$\text{Log} (3/3) = 0$	0	0	0
Belle	$\frac{1}{4}$	$\frac{1}{6}$	$\frac{0}{4}$	$\text{Log} (3/2) = 0,17$	0,04	0,028	0
Mais	$\frac{0}{4}$	$\frac{1}{6}$	$\frac{0}{4}$	$\text{Log} (3/1) = 0,47$	0	0,078	0
Difficile	$\frac{0}{4}$	$\frac{1}{6}$	$\frac{0}{4}$	$\text{Log} (3/1) = 0,47$	0	0,078	0
Joyeuse	$\frac{0}{4}$	$\frac{0}{6}$	$\frac{1}{4}$	$\text{Log} (3/1) = 0,47$	0	0	0,117

Nous avons le même inconvénient que le Bag of words sur le sens contextuel des mots. Cependant nous limitons notre volume du vecteur avec des termes présents partout qui peuvent être supprimés.

C'est cette deuxième approche que j'ai conservée pour faire ma vectorisation.



### 3) Modélisation

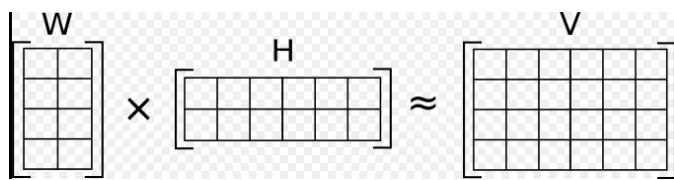
#### 1) APPROCHE NON SUPERVISEE

Dans un premier temps il m'a été demandé de réaliser une approche non supervisée afin de créer des regroupements de questions. Il s'agit de trouver des Topics qui peuvent permettre de classer nos questions. Ce nombre de Topics devra être challengé afin de trouver le nombre optimal.

Pour cela je vais utiliser des algorithmes de Topic Modelling tel que NMF (Non-negative matrix factorization) ou LDA (Latent Dirichlet Allocation).

#### NMF

La matrice de départ avec un nombre important de valeurs nulles va être décomposée en deux autres matrices W et H.  $V = W * H$


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

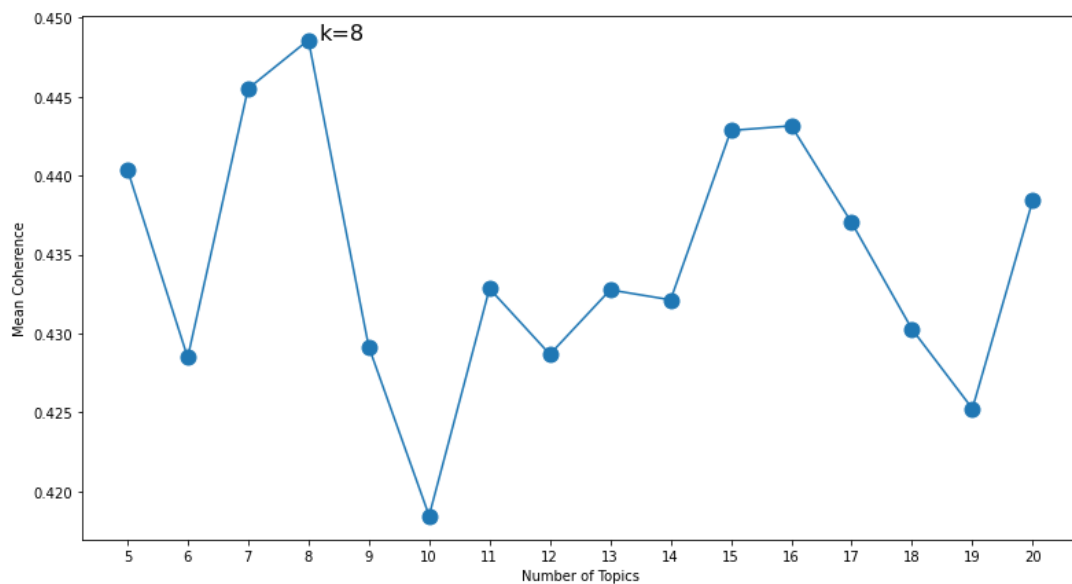
Ainsi la matrice W représente nos différents Topics alors que la matrice H l'importance des mots dans ce Topic.

#### Sélection du nombre de TOPIC

Un point important de l'approche NMF est le choix du nombre de TOPIC. Afin de déterminer ce nombre de Topics j'ai calculé les matrices W et H pour un nombre de topic allant de 5 à 20.

J'ai stocké ces résultats puis à l'aide de la librairie SPACY j'ai calculé la similarité des paires de mots contenus dans un même topic. Enfin j'ai calculé la cohérence globale de ces différents Topics pour avoir un score global. Ceci me permet de challenger mon nombre de Topic et de trouver le chiffre optimal.

J'ai affiché une courbe afin de présenter le résultat



J'obtiens un nombre de TOPIC optimal à 8 qui me permet de relance mon traitement avec un K=8 afin d'analyser plus précisément mes deux matrices W et H.

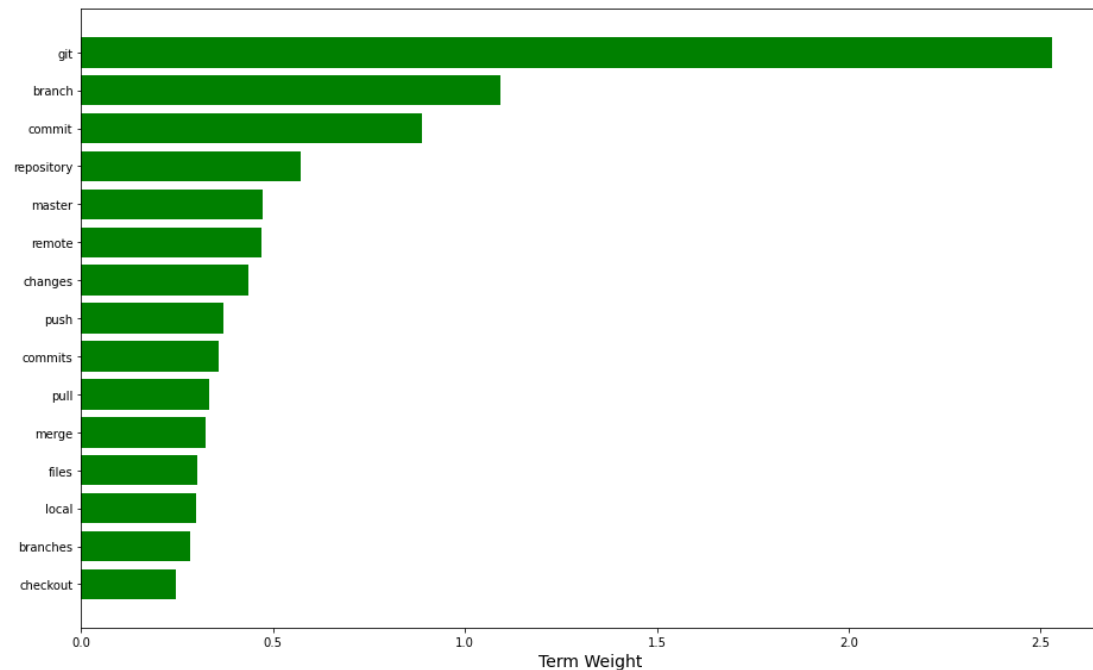
### Voici les mots les plus représentés par TOPIC

Topic 01: class, function, use, code, public, return, method, new, difference, object  
 Topic 02: git, branch, commit, repository, master, remote, changes, push, commits, pull  
 Topic 03: string, convert, character, characters, java, strings, split, replace, str, want  
 Topic 04: div, c#, element, jquery, html, css, height, width, text, input  
 Topic 05: file, line, files, command, directory, script, run, error, path, bash  
 Topic 06: array, javascript, object, var, element, elements, convert, numpy, way, value  
 Topic 07: table, sql, select, mysql, column, database, server, query, data, value  
 Topic 08: list, python, way, x, lists, dictionary, c#, get, item, like

Ce qui me permet de définir les titres suivants :

Topic 1	Exemple de code
Topic 2	GIT
Topic 3	Traitement des chaines de caractères
Topic 4	Language Frontend (C#,HTML, CSS)
Topic 5	Shell
Topic 6	Gestion des tableaux
Topic 7	Base de données (SQL, MySQL)
Topic 8	Python

Plus précisément si je regarde le TOPIC 2



Le résultat est pas mal du tout !

## LDA

Il s'agit d'une méthode probabiliste qui va chercher à déterminer la probabilité d'appartenance d'un texte à un topic en fonction des mots présents dans le texte.

Avant de réaliser le LDA, une vectorisation avec un Bag of Words a été réalisée.

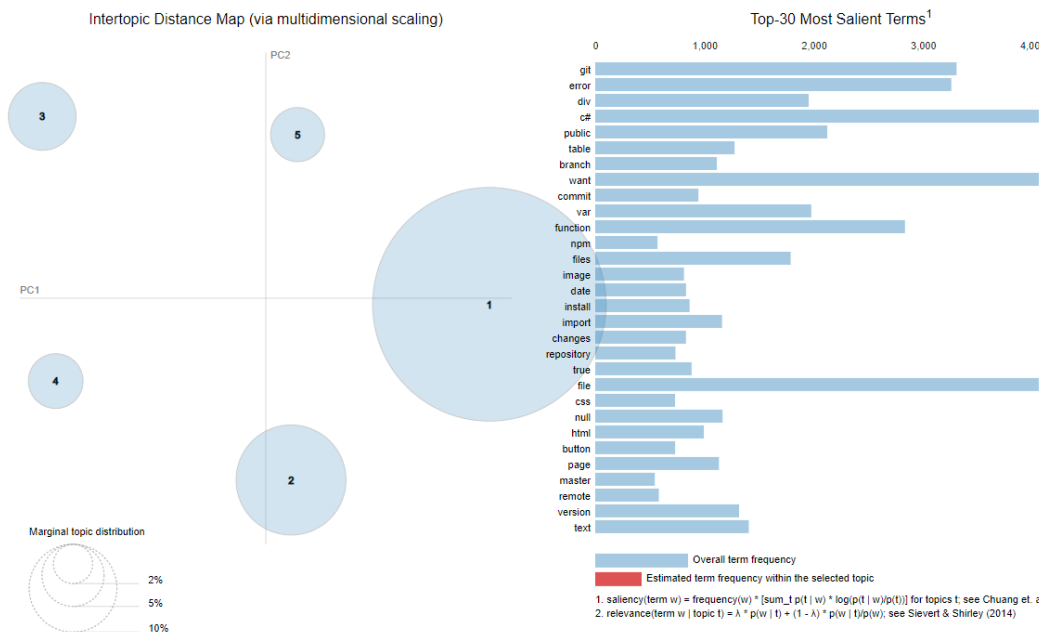
Puis afin de définir les hyper paramètres optimaux j'ai lancé une validation croisée.

Voici les paramètres optimaux obtenus :

learning\_decay': 0.9, 'learning\_method': 'online', 'n\_components': 5

Ici cinq topics ont été détectés. J'ai une préférence pour NMF qui me semble coller davantage à mon intuition.

## Et la représentation des différents TOPICS



## 2) APPROCHE SUPERVISEE

La classification que l'on doit effectuer est une classification Multi Label, c'est-à-dire qu'une question peut se voir attribuer plusieurs étiquettes (Tags) en fonction de son contenu. Pour cela j'ai utilisé la librairie MultiLabelBinarizer de Scikit learn.

Pour cela il a donc fallu définir les métriques qui me permettraient d'évaluer mes différents modèles. Après une lecture attentive de différents articles je me suis arrêté sur les éléments suivants :

- Hamming loss : Représente les mauvaises classifications (Fraction moyenne d'étiquettes incorrectes) (Score parfait 0)
- Accuracy : Précision du modèle en moyennant chaque catégorie
- F1-score : Balance entre précision et rappel. La précision et le rappel sont calculés à partir de la matrice de confusion.
- Jaccard : Taille des intersections des étiquettes prédites et des étiquettes vraies divisée par la taille de l'union des étiquettes prédites et vraies. (1 score parfait)

Plusieurs modèles ont été réalisés :

- Naive Bayse après TF-IDF
- Arbre de décision après NMF
- Arbre de décision sans NMF
- Arbre de décision après Truncate SVD
- Arbre de décision après ACP

## Synthèse des résultats

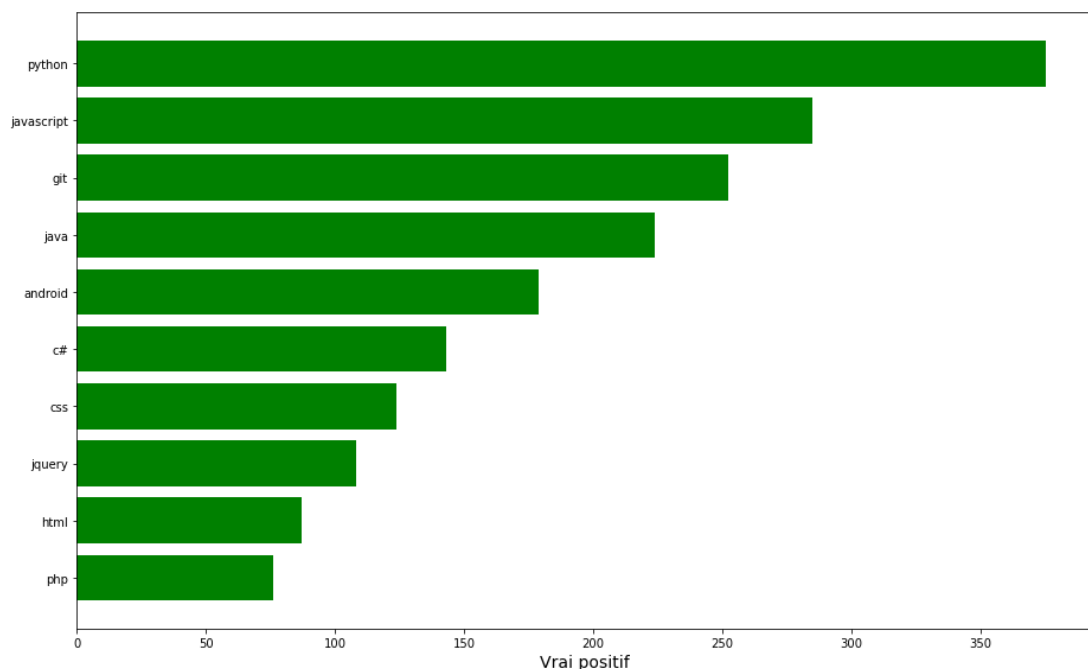
Modèle	F1 Score	Jaccard
Naive Bayse après TF-IDF	0,084	0,044
Arbre de décision après NMF	0,112	0,059
Arbre de décision après TF-IDF	0,573	0,402
Arbre de décision après Truncate SVD	0,350	0,212
Arbre de décision après ACP	0,348	0,210

Le modèle arbre de décision après TF-IDF semble le modèle le meilleur modèle. C'est celui que je vais retenir.

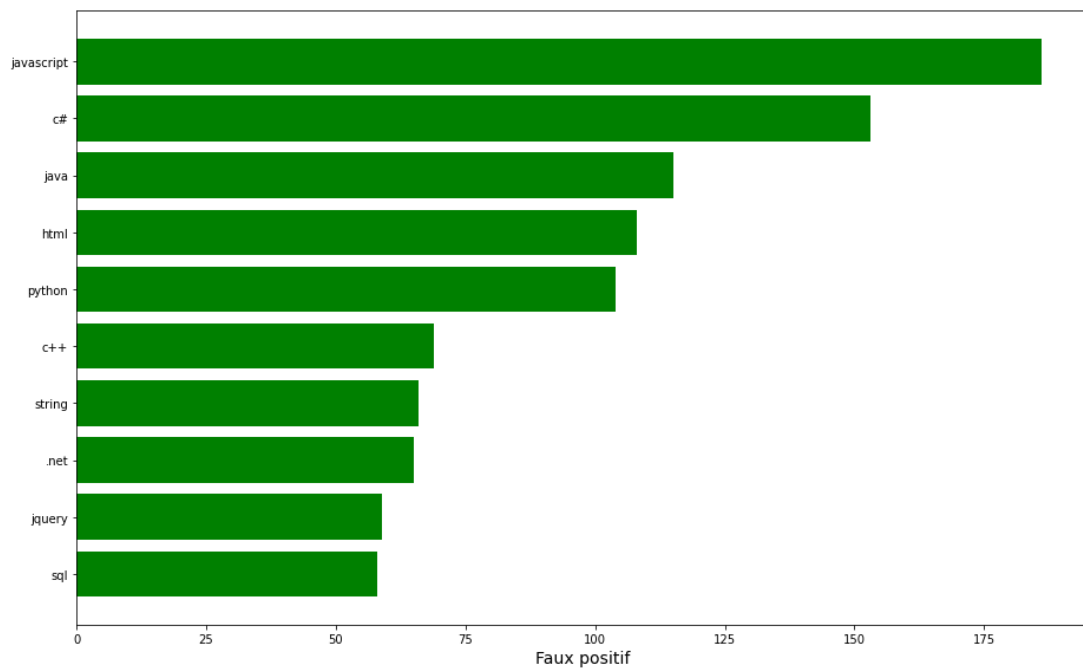
## Analyse des prédictions

J'ai regardé plus particulièrement les prédictions afin de voir si certains termes étaient très mal prédits.

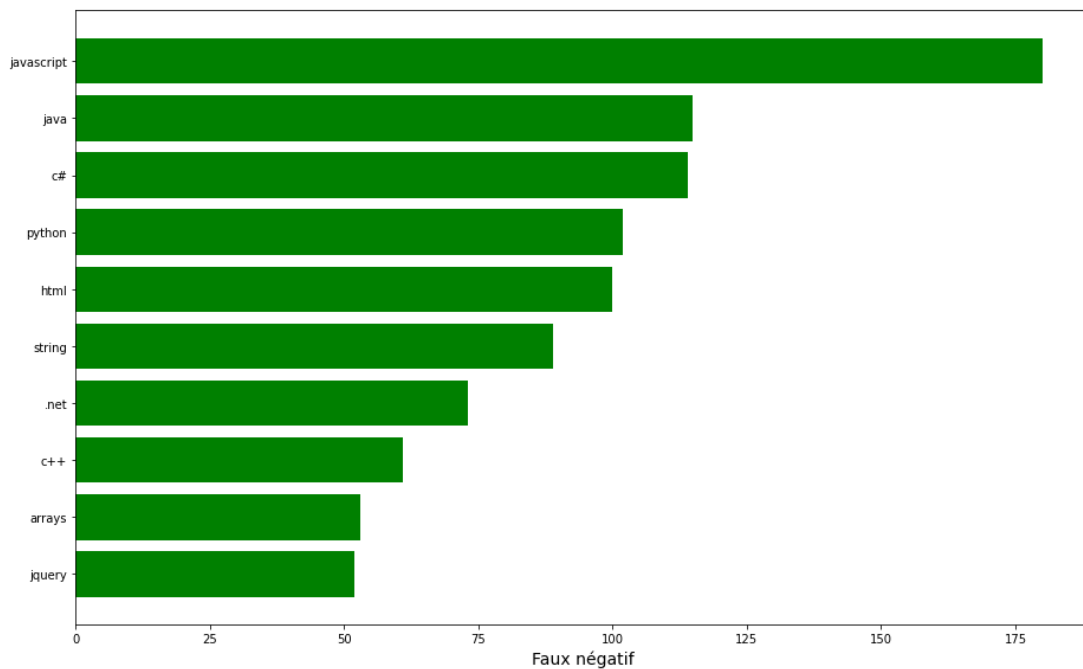
Le premier graphique présente les termes bien prédits. Python sort très loin devant suivi par Javascript.



En regardant les termes prédits à tort on constate que Javascript est très loin devant. Ceci relativise du coup les bonnes prédictions de ce terme.



Enfin on constate aussi ceux qui n'ont pas été bien prédits. Javascript est toujours en tête d'affiche.



Ces éléments permettent une première analyse rapide. Il faudrait creuser davantage ces analyses afin d'améliorer la prédiction de certains termes.

## 4) Déploiement API

Pour le déploiement au niveau de l'API j'ai choisi le modèle Arbre de décision sans NMF donc à partir des données après TF-IDF (Aucune réduction de dimension). C'est celui qui donnait les meilleurs résultats.

J'ai utilisé Flask comme framework pour construire mon application qui est constituée des éléments suivants :

Model :

decisionTreeClassifier.pickle (Modèle prédictif)

multilabel.pickle (Tags de résultat à prédire en multi label)

vectorizer.pickle (traitement TF-IDF de vectorisation des données)

templates :

index.html (page principale pour rentrer son titre et son corps de texte)

predict.html (page de prédiction des tags)

static / css / main.css (CSS pour gérer l'affichage)

P6\_Deploiement.py route flask d'exécution

Util.py librairie utilitaire de traitement

Le déploiement a été réalisé sur Heroku à l'adresse suivante :

<https://predictiontags.herokuapp.com/>

## 5) Git

Mon repository Git Hub se trouve à l'adresse suivante :

[https://github.com/jeje0410/P6\\_Categorisation\\_questions](https://github.com/jeje0410/P6_Categorisation_questions)

## 6) Conclusion

Ce projet m'a permis d'appréhender énormément d'éléments différents. Le traitement du langage naturel, la réduction de dimension avec NMF, la mise en place d'un Github et le déploiement d'un API Flask.

Le traitement du langage naturel et sa vectorisation m'ont particulièrement plu.

L'approche non supervisée avec la catégorisation des questions m'a impressionné. J'ai trouvé les résultats particulièrement pertinents. Je reste sur ma faim sur la partie

supervisée avec des résultats décevants. J'ai tenté une analyse des prédictions afin d'identifier des axes d'amélioration. Il faudrait poursuivre cette analyse.