



# Projet n° 5 : Catégorisez automatiquement des questions

Céline Mendola

**01**

## **Introduction**

---

Présentation du problème, présentation des données

**02**

## **Analyses**

---

Nettoyage et analyses des données textuelles

**03**

## **Modélisations**

---

Modèles non supervisés et supervisés

**04**

## **Conclusion**

---

Choix du modèle, déploiement d'une app avec flask et heroku



# Introduction

---

# Présentation de la problématique

- On souhaite développer un système de suggestion de tags pour le célèbre site stackoverflow. A l'aide d'algorithmes de machine learning, il faudra assigner des tags pertinents à une question posée sur le site.
- Les données sont récupérées grâce à l'outil stackexchange explorer. On récupère tous les posts avec titres créés après 2011, avec leurs scores, leurs nombres de vues et de réponses sur les site.

02

# Analyses

---

## Nettoyage des données

```
df.Score.describe()
```

count	50000.000000
mean	20.709520
std	50.833301
min	6.000000
25%	7.000000
50%	10.000000
75%	17.000000
max	2554.000000
Name: Score, dtype: float64	

Pour garder les posts les plus pertinents, nous filtrons sur les scores  $>30$ . Nous obtenons un dataset de 6 500 observations environ.

# Nettoyage des données : Posts et titres

```
<p>There are a lot of differences between C and C++ and came to stuck on one of them \nThe same code gives an error in C while just executes fine in C++ \nPlease explain the reason </p>\n\n<pre><code>int main(void)\n{\n  int a=10,b;\n  a&gt;=5?b=100:b=200;\n}\n</code></pre>\n\n<p>The above code gives an error in <strong>C</strong> stating <strong>lvalue</strong> required while the same code compiles fine in <strong>C++</strong></p>\n
```

## ➤ Mettre le texte en minuscule

```
text = text.lower()
```

## ➤ Enlever les balises, les tags, les retours à la ligne, les chiffres, les urls

```
text = re.sub(r'<[^>]+>|\n|\d+|\t|\s{2,}|\&[gl][te]|\r|\f|http.+?(?="|<)', ' ', text)
```

## ➤ Enlever la ponctuation

```
text = re.sub(r'[\'\.\-!\"#$%&\*,:;=<=>?@^`()~={}\[\]\[\]\[\]', ' ', text)
```

## ➤ Enlever les mots à une lettre sauf “r” et “c”

```
text = re.sub(r'\b[abcdefghijklmnopqrstuvwxyz]\b', ' ', text)
```

# Nettoyage des données : Posts et titres

<p>There are a lot of differences between C and C++ and came to stuck on one of them \nThe same code gives an error in C while just executes fine in C++ \nPlease explain the reason </p>\n\n<pre>  
<code>int main(void)\n{\n int a=10,b;\n a&gt;=5?b=100:b=200;\n}\n</code></pre>\n\n<p>The above code gives an error in <strong>C</strong> stating <strong>lvalue</strong> required while the same code compiles fine in <strong>C++</strong></p>\n

## ➤ Enlever les mots à deux lettres

```
text = re.sub( r'\b[a-z][a-z]\b',' ', text)
```

## ➤ Finalement enlever les tabulations et espaces multiples.

```
text = re.sub(r'\s{2,}|\t',' ',text)
```

## ➤ Suppression des stopwords



# Nettoyage des données : les stop words

Stopwords : mots qui n'apportent pas beaucoup d'information.

On a utilisé la liste des stopwords de la librairie nltk à laquelle nous avons ajouté au fur et à mesure certains tokens comme 'like', 'thank', 'use', 'quot', 'try', 'know', 'find', 'way' ...

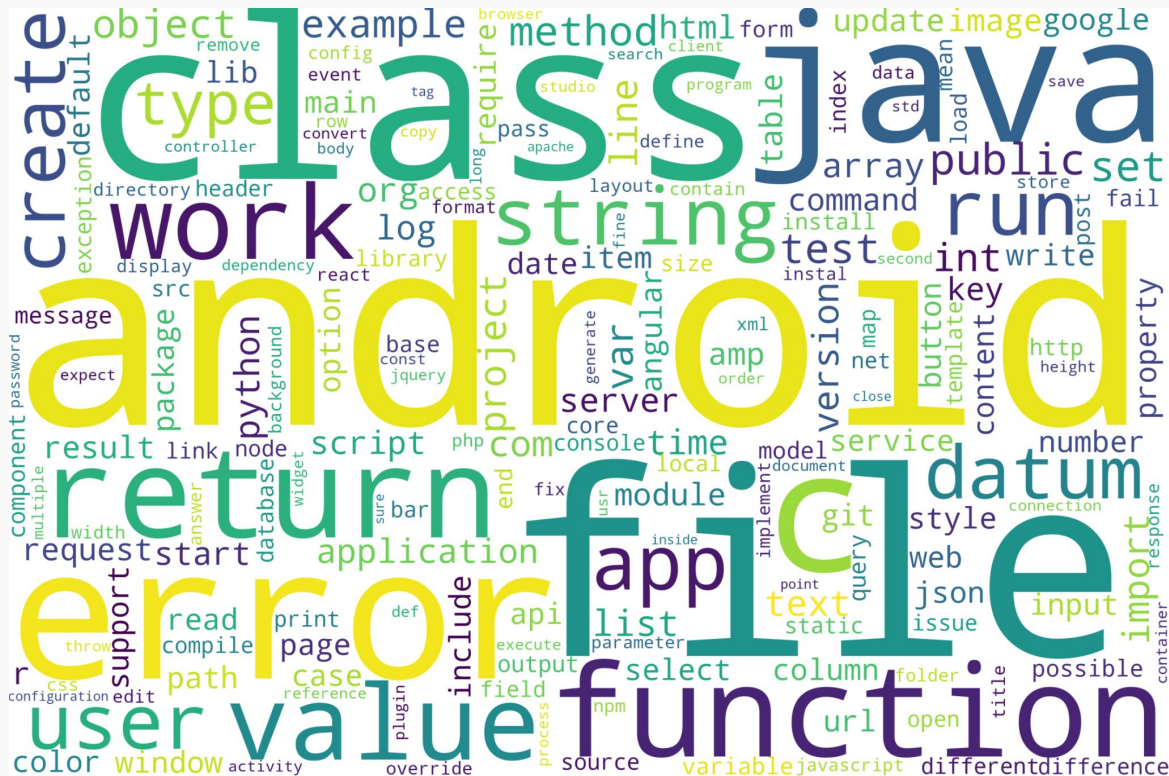
# Nettoyage des données : lemmatization

But : utiliser un seul token pour les différentes déclinaisons d'un même mot (pluriel/ conjugaison/participes ...)

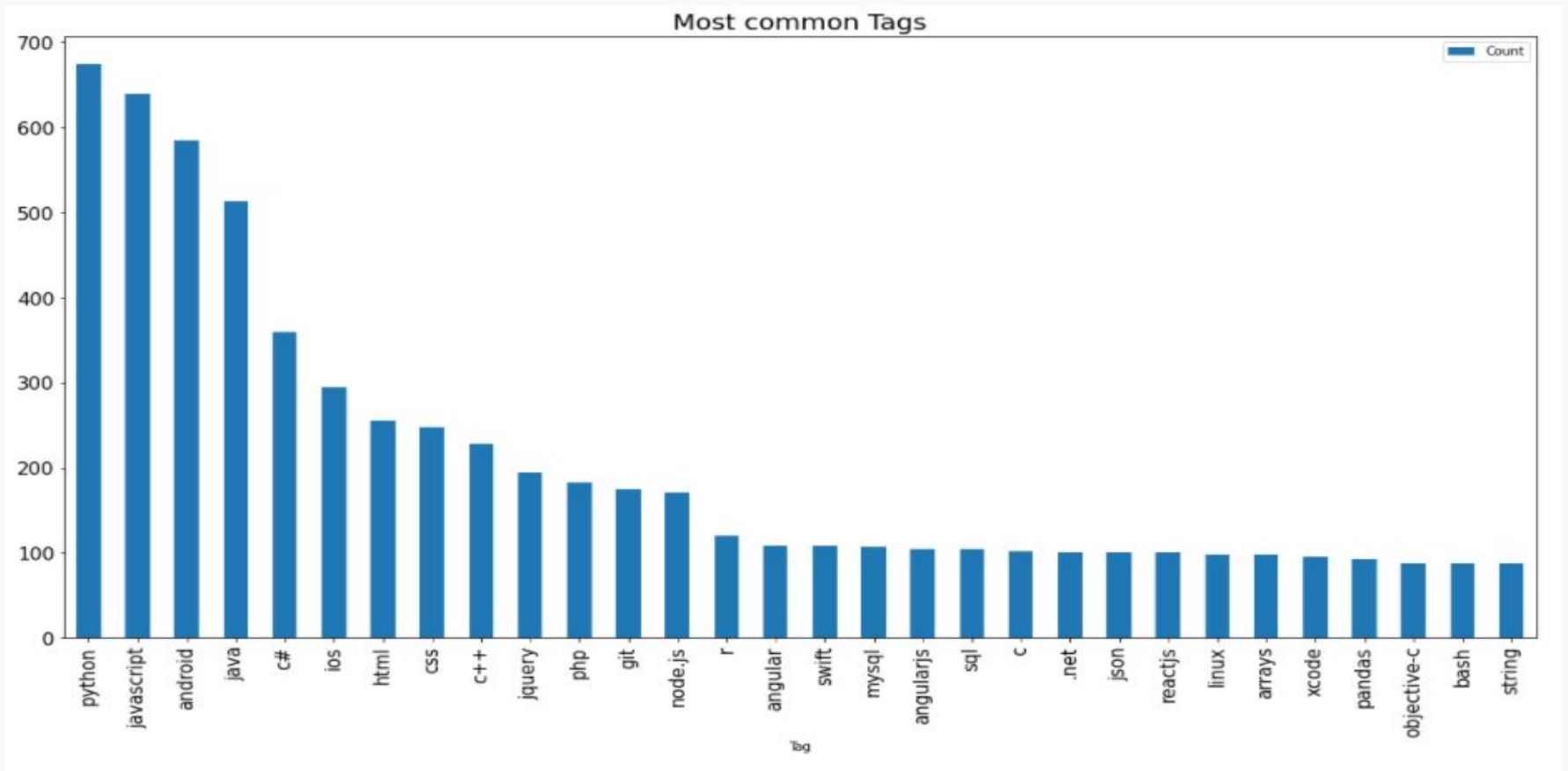


Utilisation dans le projet de la lemmatization (méthode `.lemma_` de la librairie `spacy`)

# Analyses des mots présents dans les posts et titres



# Analyses des Tags les plus présents



# Preprocessing

- Utilisation des bag of words : technique permettant de vectoriser le corpus de texte.
- Une 1ère méthode : compter le nombre d'occurrences de chaque mot dans chaque document. ( “ Term frequency”)

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

BOW on Surfin' Bird

# Preprocessing

- Une 2ème méthode : multiplier les term frequency par les “inverse term frequency”

$tf(t, d) = |\text{Number of times term } t \text{ appears in document } d|$

$$idf(t, D) = \frac{|\text{Number of documents}|}{|\text{number of documents that contain term } t|}$$
$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

- $t$  is the word or token.
- $d$  is the document.
- $D$  is the set of documents in the corpus.

- Avantage : Donner plus poids aux tokens qui apparaissent dans peu de documents et moins aux mots très fréquents.
- D'autres méthodes de vectorisation existent. On utilise en input de nos algorithmes la vectorisation TFIDF.

# 03 **Modélisations**

---

# Modélisation de sujets : Principe

La modélisation automatique de sujet permet de détecter les sujets latents abordés dans un corpus de documents.

Nous appliquerons deux modèles non supervisés: LDA et NMF.



# Modélisation de sujets : LDA

Le modèle LDA (Latent Dirichlet Allocation) est un modèle probabiliste génératif dans lequel chaque document du corpus est associé à une distribution de différents sujets. Également, chaque sujet est associé à une distribution de mots présents dans le corpus.

# Modélisation de sujets : LDA

## Principales étapes :

- On fixe le nombre  $K$  de thèmes. On initialise chaque mot à l'un des  $K$  thèmes.
- Pour chaque document  $d$ , chaque mot  $w$  du document  $d$  et chaque sujet  $t$ , on calcule :

$P(\text{sujet } t \mid \text{document } d)$  la proportion de mots du document  $d$  associé au sujet  $t$

$P(\text{mot } w \mid \text{sujet } t)$  la proportion d'affectation au sujet  $t$  dans l'ensemble du corpus qui proviennent du mot  $w$ . On affecte alors un nouveau sujet  $T'$  au mot  $w$  avec la probabilité

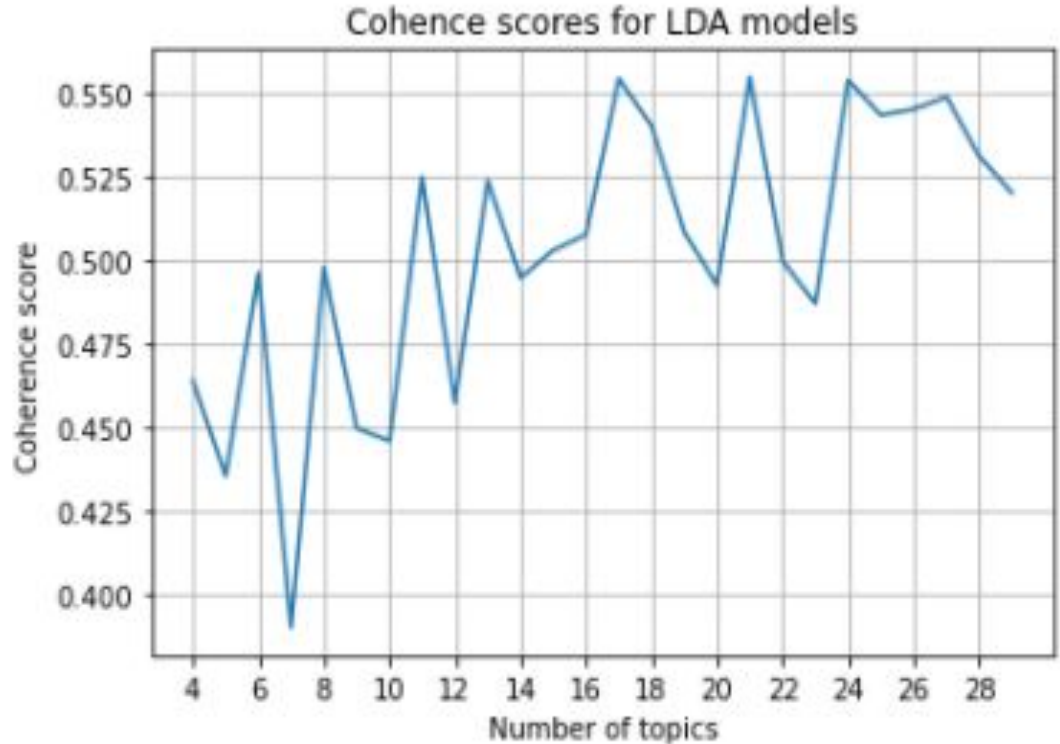
$$P(\text{sujet } T' \mid \text{document } d) * P(\text{mot } w \mid \text{sujet } T')$$

On réitère les dernière étape, les affectations se stabilisent.

# Modélisations de sujets : LDA

On utilise le score de cohérence pour trouver le nombre de sujets.

Ici, on le fixera à 17



# Modélisations de sujets : LDA

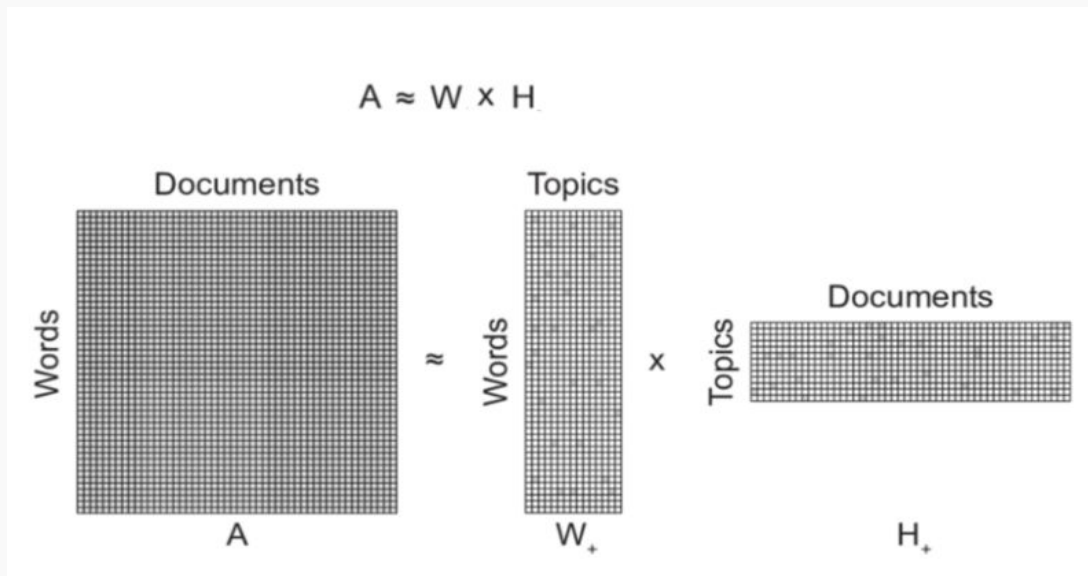
```
out[]: [(0,
  '0.031*job" + 0.024*flask" + 0.024*blank" + 0.021*alpha" +
  0.021*unique"
  (1,
    '0.029"dataframe" + 0.029*panda" + 0.028*csv" + 0.028*column" +
    0.026*cell"
  (2,
    '0.030*table" + 0.026*sql" + 0.025*query" + 0.023*mysql" +
    0.017*column"
  (3,
    '0.026*println" + 0.025*modal" + 0.024*integer" + 0.020*decimal"
  (4,
    '0.022*session" + 0.018*route" + 0.017*placeholder" +
    0.015*angular"
  (5,
    '0.022*git" + 0.012*file" + 0.011*install" + 0.011*npm" +
    0.011*branch"
  + 0.010*run" + 0.010*command" + 0.009*commit" + 0.009*project" +
    0.008*master)'),
  (6,
    '0.036*docker" + 0.024*notification" + 0.022*listview" +
    0.019*mongodb" + 0.018*architecture"
  (7,
    '0.016*string" + 0.014*value" + 0.013*array" + 0.012*c" +
    0.012*int" + 0.012*list"
  (8,
    '0.028*age" + 0.023*regex" + 0.023*history" + 0.019*checkbox" +
    0.017*uuid" + 0.014*haskell" + 0.013*clause" + 0.013*slash" +
    0.012*slot" + 0.012*condition'),
```

```
(9,
  '0.038*laravel" + 0.038*symbol" + 0.022*blue" +
  0.021*employee" + 0.017*linearlayout" '),
  (10,
    '0.042*android" + 0.029*java" + 0.020*eclipse" +
    0.018*gradle" + 0.015*std" +,
  (11,
    '0.009*file" + 0.007*user" + 0.006*app" + 0.006*class" +
    0.006*error" +
  (12,
    '0.025*domain" + 0.022*iframe" + 0.021*programmatically" +
    0.017*dict" + 0.016*xcode" ,
  (13,
    '0.024*area" + 0.022*preference" + 0.020*svn" +
    0.018*textbox" + 0.015*occurrence",
  (14,
    '0.026*android" + 0.015*color" + 0.011*background" +
    0.010*width" + 0.009*height"
  (15,
    '0.033*byte" + 0.029*col" + 0.028*char" + 0.025*random" +
    0.024*timestamp"
  (16,
    '0.021*spring" + 0.020*bean" + 0.020*vim" + 0.020*video"
  + 0.017*svg"
```

Tests de prédiction dans test.xlsx

# Modélisations de sujets : NMF

Le modèle NMF (non-negative matrix factorization) est une méthode d'algèbre linéaire qui peut être utilisée pour la prédiction de sujets.



$A$ ,  $W$  et  $H$  sont à termes positifs.

# Modélisations de sujets : NMF

Topic 0:

error user run request server test web api

Topic 1:

android studio layout\_width com layout\_height

Topic 2:

file directory folder line project path open txt

Topic 3:

string convert object json public stre format str

Topic 4:

git branch commit master push repository github

Topic 5:

table sql query key select database mysql create

Topic 6:

class public static method type extend object href

Topic 7:

function return var console datum javascript log

Topic 8:

date day datetime month format year time hour convert

Topic 9:

python line module print import package install pip lib

Topic 10:

image background color css text width button html

Topic 11:

array numpy object arr php convert index return byte

Topic 12:

list item loop collection index contain convert object

Topic 13:

value select option type key input property default

Topic 14:

int std struct amp cout main char include return type

Topic 15:

java org lang method eclipse androidruntime exception

Topic 16:

column dataframe panda datum row frame index csv col

Topic 17:

app io xcode component react play device google angular

# Classification multi-étiquettes

Plusieurs tags peuvent être associés à un même post, nous sommes donc face à un problème de classification binaire. Plusieurs méthodes sont possibles :

- appliquer un modèle déjà adapté
- Utiliser un même classificateur binaire pour chaque étiquette, ce qu'on appelle la BinaryRelevance :

X	Y <sub>1</sub>	Y <sub>2</sub>	Y <sub>3</sub>	Y <sub>4</sub>
x <sup>(1)</sup>	0	1	1	0
x <sup>(2)</sup>	1	0	0	0
x <sup>(3)</sup>	0	1	0	0
x <sup>(4)</sup>	1	0	0	1
x <sup>(5)</sup>	0	0	0	1



X	Y <sub>1</sub>	X	Y <sub>2</sub>	X	Y <sub>3</sub>	X	Y <sub>4</sub>
x <sup>(1)</sup>	0	x <sup>(1)</sup>	1	x <sup>(1)</sup>	1	x <sup>(1)</sup>	0
x <sup>(2)</sup>	1	x <sup>(2)</sup>	0	x <sup>(2)</sup>	0	x <sup>(2)</sup>	0
x <sup>(3)</sup>	0	x <sup>(3)</sup>	1	x <sup>(3)</sup>	0	x <sup>(3)</sup>	0
x <sup>(4)</sup>	1	x <sup>(4)</sup>	0	x <sup>(4)</sup>	0	x <sup>(4)</sup>	1
x <sup>(5)</sup>	0	x <sup>(5)</sup>	0	x <sup>(5)</sup>	0	x <sup>(5)</sup>	1

On ne prend pas en compte les tags qui n'apparaissent que dans au maximum 1% des posts.

# Classification multi-étiquettes

	macro_f1	micro_f1	accuracy
<b>KNN</b>	0.490787	0.557951	0.353955
<b>Naive Bayes</b>	0.116853	0.294155	0.160243
<b>Linear SVC</b>	0.608758	0.689837	0.451318

Choix de la SVC linéaire.

Ajout d'une ACP dans le pre-processing : performances un peu moins bonnes mais temps de prédiction + rapide.





04

# Conclusion

---

# Conclusion

Choix d'un modèle supervisé avec le modèle SVC.

Déploiement de l'app avec flask et heroku