# Projet n°5 :

# Catégorisez automatiquement des questions

Parcours ingénieur machine learning OPENCLASSROOMS

Céline Mendola

# Introduction :

Le but de ce projet est de développer un système de suggestion de tags pour le célèbre site stackoverflow. Il s'agit d'un site de questions-réponses sur le développement informatique.
A partir d'une base de données de l'ensemble des posts du site, nous analyserons dans un premier temps les données textuelles dont nous disposons. Puis nous mettrons en oeuvre des méthodes de Natural language processing afin d'élaborer nos modèles. Après pre-processing du corpus de texte, nous appliquerons deux grandes familles de modèles. D'une part, les modélisations de sujets qui sont des méthodes non supervisées. D'autre part, la classification multi-étiquettes pour les méthodes supervisées.
Le modèle le plus pertinent sera déployé à l'aide de flask et heroku.
Dans la suite du rapport:

- les tokens correspondront aux mots
- les documents correspondront aux posts
- le corpus du texte correspondra à l'ensemble des posts

# I. Exploration des données

## 1. Import du dataset

Nous importons le dataset de stackenchange explorer. La requête utilisée est la suivante :

```
Select CreationDate, Score, ViewCount, Body, Title, Tags,
AnswerCount
from Posts
where Score > 5
      and (Title is not null)
      and year(CreationDate) > 2011
```

Voici les premières lignes de la table obtenue:

| | CreationDate | Score | ViewCount | Body | Title | Tags | AnswerCount |
|---|---|---|---|---|---|---|---|
| 0 | 2018-07-05 18:47:59 | 6 | 712 | <p>I've used <code>select_for_update() </code> ... | Django: select_for_update() on Foreign Key Rel... | <django><django-models><django-queryset> | 1 |
| 1 | 2013-11-03 22:58:08 | 9 | 1761 | <p>I'm using the filter function in Python (3.... | Listing a filter object twice will return a bl... | <python><python-3.x><iterator> | 1 |
| 2 | 2013-11-03 22:58:22 | 6 | 826 | <p>While learning how to create Lua file outpu... | Can I save into a specific folder? | <file-io><lua><directory><love2d> | 1 |
| 3 | 2013-11-03 23:00:07 | 27 | 25955 | <p>I had a .profile file that I was reading an... | .profile not working from terminal in mac | <macos><bash><terminal><alias><.profile> | 6 |
| 4 | 2015-02-05 05:52:03 | 10 | 975 | <p>There are a lot of differences between C an... | Ternary operator in C vs C++ | <c++><c><ternary-operator><conditional-operator> | 3 |

## 2. Premier filtre sur les scores

```
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.

## 3. Nettoyage des données des posts et des titres

Dans les posts et les titres, nous avons la présence de balises, de ponctuation, d'urls, d'espaces multiples… Nous définissons la fonction `clean_text` suivante que nous appliquerons aux variables Body et Title.

```
def clean_text(text):
    '''first cleaning of text'''
    #Put text in lowercase
    text = text.lower()

    #Remove tags, back to line, digits, urls, urls, double spaces, tabs
    text = re.sub(r'<[^<>]+>|\n|\d+|\t|\s{2,}|&[gl][te]|\r|\f|http.+?(?="|<)',
                  ' ',
                  text)
    #Remove punctuation
    text = re.sub(r'[\'\.\-!"#$%&\\*,:;⟺?@^`()|~={}\/\[\]\□',
                  ' ',
                  text)
    #Remove isolated digits
    text = re.sub(r'\b[0-9]+\b',
                  ' ',
                  text)
    #Remove isolated letters except "r" and "c" for programming languages
    text = re.sub( r'\b[abdefghijklmnopqstuvwxyz]\b',
                  ' ',
                  text)
    # Remove two letter words
    text = re.sub( r'\b[a-z][a-z]\b',
                  ' ',
                  text)
    #Remove double spaces and tabs
    text = re.sub(r'\s{2,}|\t',
                  ' ',
                  text)
    text = text.strip()

    return text
```

Ensuite, nous enlevons les “stop words” qui n’apportent pas beaucoup d’information pour l’application de nos modèles. Nous utilisons la liste de stopwords du package nltk que nous étendons avec certains mots comme : 'like', 'thank', 'use', 'quot', 'try', 'know', 'find', 'way' … Cette liste a été ajustée au fur et à mesure des analyses.

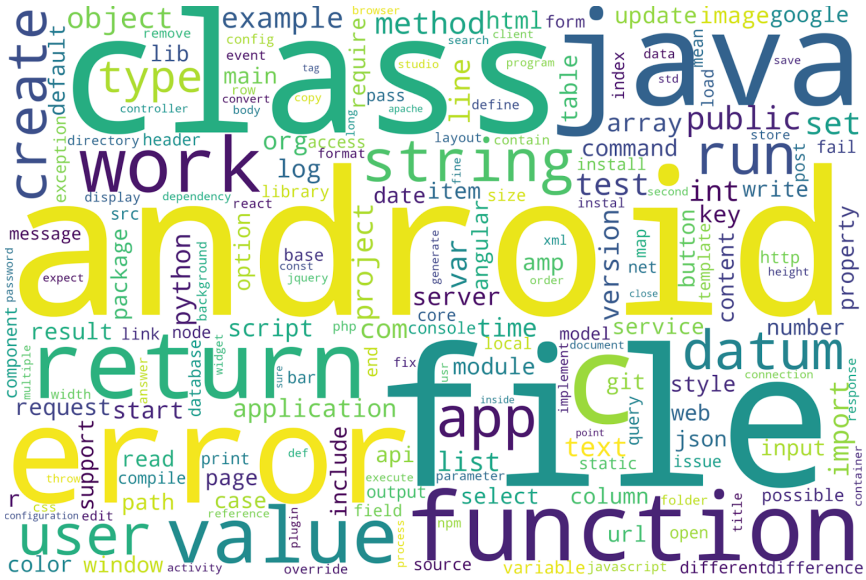
Remarquons que les mots à une lettre (à part r et c) et à deux lettres ont été supprimés avec l’application de la fonction clean_text.

Enfin, on lemmatize les mots du texte en utilisant la librairie spacy.

Exemple :

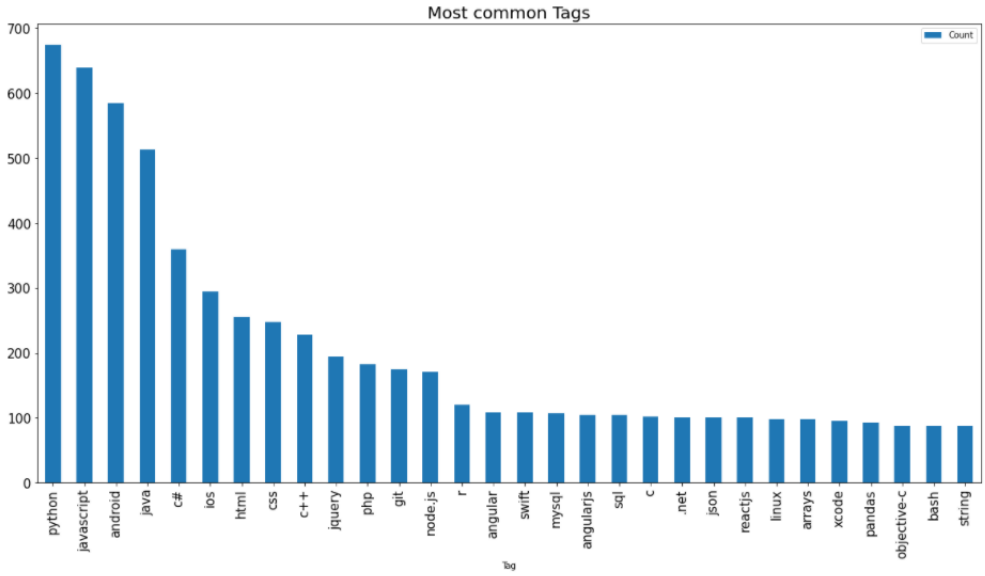
```
In[] : lemmatize(remove_stop_words("""I need a good explanation (references are a plus) on Python's slice notation.
To me, this notation needs a bit of picking up.
It looks extremely powerful, but I haven't quite got my head around it.""")) 

Out [] : 'explanation reference plus Python slice notation notation need bit pick look extremely powerful get head'
```

Nous concaténons les corps et titres du post dans une variable *Preprocessed_text* qui sera l’input pour réaliser les bag of words de nos futurs modèles. On présente ci-dessous un nuage de mots de notre nouvelle variable. Les mots les plus présents sont “android”, “file”, “class”  et “java”




## 4. Analyse des tags

Comme le montre le graphique ci-dessous, Python, javascript et android sont les tags les plus fréquents dans notre dataset.




## 5. Pre-processing

Le Bag of words est une technique permettant de vectoriser le corpus de texte afin de le donner en entrée de nos algorithmes.

- Une méthode consiste à calculer le nombre d'occurrences des mots 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

- Une seconde méthode consiste à multiplier les term frequency par les “inverse term frequency”).

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

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

$$tfidf(t,d,D) = tf(t,d).idf(t,D)$$

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

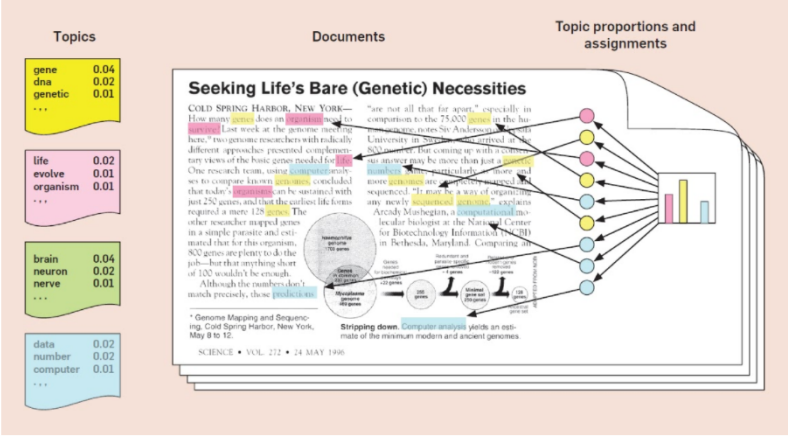
L’avantage de cette vectorisation est de donner plus de poids aux tokens qui apparaissent dans peu de documents et moins de poids aux mots très fréquents.

D’autres méthodes existent, qui prennent en compte le lien sémantique des mots. Dans notre cas, nous utiliserons la vectorisation TF-IDF en entrée de nos modèles. En particulier,

# II. Prédictions des sujets des posts avec des méthodes non supervisées

## 1. Par la méthode LDA (Latent Dirichlet Allocation)

### a. Principe de l'algorithme




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

Sans entrer dans les détails mathématiques, les principales étapes de l'algorithme LDA sont les suivantes:

1) On définit un nombre K de thèmes. Pour chaque document, on initialise aléatoirement chaque mot à l'un des K thèmes..
2) Pour chaque document d, chaque mot w du document d, et chaque sujet t, on calcule :
    - P(sujet t \ document d) la proportion de mots du document d associé au sujet t
    - P(mot w \ 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(sujet T' \ document d) * P(mot w \ sujet T')

    en considérant tous les autres mots et leur affectation de thème.

3) On réitère l'étape 2) un grand nombre de fois. On atteindra alors à termes des affectations assez stables et qui font sens. On utilise alors les affectations pour déterminer la distribution des sujets dans chaque document et ainsi que celle des mots pour chaque sujet.

http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/?utm_source=pocket_mylist

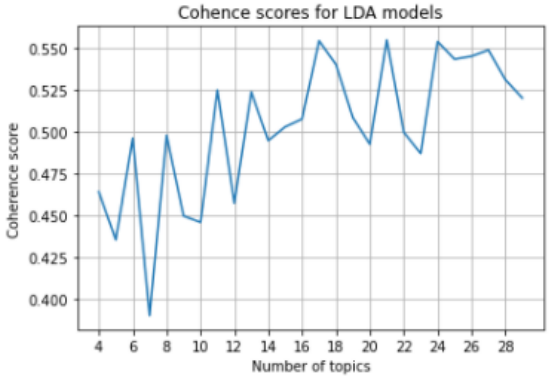
### b. Déploiement

Nous utilisons la librairie gensim.
On indexe l'ensemble des mots du corpus, en filtrant sur ceux qui n'apparaissent pas dans moins de 5 documents ou dans plus de 80 % des documents.

```
In[]: # Print dictionary
      print(dictionary.token2id)

Out[]: {'accomplish': 0, 'column': 1, 'country': 2, 'csv': 3, 'dataframe': 4, 'datum': 5, 'end': 6,
'format': 7, 'frame': 8, 'indicator': 9, 'panda': 10, 'pivot': 11, 'print': 12, 'result': 13, 'success':
14, 'table': 15, 'thought': 16, 'transpose': 17, 'value': 18, …

In[] : corpus = [dictionary.doc2bow(text) for text in data_cleaned]
         print(corpus[0])
Out[]: [(0, 1), (1, 3), (2, 3), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 3), (10, 2), (11, 2),
(12, 1), (13, 1), (14, 1), (15, 2), (16, 1), (17, 1), (18, 4), (19, 3)]
```

Dans le premier document, le mot "accomplish" apparaît une fois, "column" apparaît trois fois, "csv" apparaît trois fois etc.

Pour déterminer le nombre de thèmes t optimal, on applique l'algorithme pour t variant entre 4 et 30 et on calcule le score de cohérence : Plus il est élevé, plus le nombre de sujets est pertinent.




On fixe le nombre de sujets à 17.

```
In[]: # See the topics
lda_model.print_topics(-1,num_words=10)

Out[]: [(0,
  '0.031*"job" + 0.024*"flask" + 0.024*"blank" + 0.021*"alpha" + 0.021*"unique" + 0.018*"user_id" +
0.017*"paste" + 0.014*"reverse" + 0.014*"email" + 0.013*"alter"'),
```

```
 (1,
  '0.029*"dataframe" + 0.029*"panda" + 0.028*"csv" + 0.028*"column" + 0.026*"cell" + 0.019*"swift" +
0.018*"row" + 0.016*"func" + 0.016*"postgre" + 0.014*"excel"'),
 (2,
  '0.030*"table" + 0.026*"sql" + 0.025*"query" + 0.023*"mysql" + 0.017*"column" + 0.015*"entity" +
0.015*"select" + 0.014*"order" + 0.013*"database" + 0.013*"model"'),
 (3,
  '0.026*"println" + 0.025*"modal" + 0.024*"integer" + 0.020*"decimal" + 0.019*"split" + 0.018*"amazon" +
0.015*"double" + 0.013*"nbsp" + 0.013*"logger" + 0.013*"represent"'),
 (4,
  '0.022*"session" + 0.018*"route" + 0.017*"placeholder" + 0.015*"angular" + 0.013*"observable" +
0.013*"scope" + 0.013*"shape" + 0.012*"controller" + 0.012*"email" + 0.012*"authentication"'),
 (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" +
0.017*"webpack" + 0.016*"container" + 0.016*"exit" + 0.015*"bucket" + 0.011*"near"'),
 (7,
  '0.016*"string" + 0.014*"value" + 0.013*"array" + 0.012*"c" + 0.012*"int" + 0.012*"list" + 0.011*"date" +
0.010*"object" + 0.009*"return" + 0.008*"number"'),
 (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" +
0.015*"signal" + 0.015*"uiimage" + 0.015*"onchange" + 0.014*"xpath" + 0.014*"wordpress"'),
 (10,
  '0.042*"android" + 0.029*"java" + 0.020*"eclipse" + 0.018*"gradle" + 0.015*"std" + 0.014*"compile" +
0.011*"jar" + 0.011*"sdk" + 0.011*"studio" + 0.011*"maven"'),
 (11,
  '0.009*"file" + 0.007*"user" + 0.006*"app" + 0.006*"class" + 0.006*"error" + 0.005*"function" +
0.005*"test" + 0.005*"datum" + 0.005*"html" + 0.005*"page"'),
 (12,
  '0.025*"domain" + 0.022*"iframe" + 0.021*"programmatically" + 0.017*"dict" + 0.016*"xcode" +
0.014*"oauth" + 0.013*"watch" + 0.013*"foreign" + 0.013*"drive" + 0.012*"protocol"'),
 (13,
  '0.024*"area" + 0.022*"preference" + 0.020*"svn" + 0.018*"textbox" + 0.015*"occurrence" + 0.013*"pool" +
0.013*"viewmodel" + 0.013*"percentage" + 0.012*"touch" + 0.011*"maximum"'),
 (14,
  '0.026*"android" + 0.015*"color" + 0.011*"background" + 0.010*"width" + 0.009*"height" + 0.009*"image" +
0.008*"r" + 0.008*"activity" + 0.008*"style" + 0.008*"layout"'),
 (15,
  '0.033*"byte" + 0.029*"col" + 0.028*"char" + 0.025*"random" + 0.024*"timestamp" + 0.021*"curl" +
0.018*"buffer" + 0.016*"ssl" + 0.015*"certificate" + 0.015*"nan"'),
 (16,
  '0.021*"spring" + 0.020*"bean" + 0.020*"vim" + 0.020*"video" + 0.017*"svg" + 0.014*"scale" + 0.014*"arr"
+ 0.013*"assert" + 0.012*"uiview" + 0.011*"springframework"')]
```

On voit bien des thèmes se dégager.

### c. Prédictions

On définit la fonction de prédiction: pour définir les tags d'un text, on définit le mot le plus important du topic prédit, puis les mots qui apparaissent dans le topic prédit et dans le texte.

Exemple 1 : Lien du post
https://stackoverflow.com/questions/273192/how-can-i-safely-create-a-nested-directory

```
In[] : text1 = """How can I safely create a nested directory?
        What is the most elegant way to check if the directory a file is going to be written to exists,
        and if not, create the directory using Python? Here is what I tried:
        import os\ file_path ="/my/directory/filename.txt"
        directory = os.path.dirname(file_path)
        try:
        os.stat(directory)
        except:
        os.mkdir(directory)
        f = file(filename)
        Somehow, I missed os.path.exists (thanks kanja, Blair, and Douglas). This is what I have now:
        def ensure_dir(file_path):
        directory = os.path.dirname(file_path)
        if not os.path.exists(directory):
        os.makedirs(directory)
        Is there a flag for open(), that makes this happen automatically?"""

print(predict_lda(text1))

Out[] : ['git', 'directory', 'file', 'path']
```

Exemple 2 : Lien du post
https://stackoverflow.com/questions/509211/understanding-slice-notation

```
In[] : text2 = """I need a good explanation (references are a plus) on Python's slice notation.
To me, this notation needs a bit of picking up.
It looks extremely powerful, but I haven't quite got my head around it."""
print(predict_lda(text2))

Out[] : ['python', 'string']
```

Exemple 3 : Lien du post
:https://stackoverflow.com/questions/1642028/what-is-the-operator-in-c-c

```
In[] : text3 = """After reading Hidden Features and Dark Corners of C++/STL on comp.lang.c++.moderated, I
was completely surprised that the following snippet compiled and worked in both Visual Studio 2008 and G++
4.4. Here's the code:
#include <stdio.h>
int main()
{
    int x = 10;
    while (x ⟶ 0) // x goes to 0
    {
        printf("%d ", x);
    }
}
Output:
9 8 7 6 5 4 3 2 1 0
I'd assume this is C, since it works in GCC as well. Where is this defined in the standard, and where has
it come from?"""


print(predict_lda(text3))

Out[]: ['c', 'string', 'c++', 'int']
```

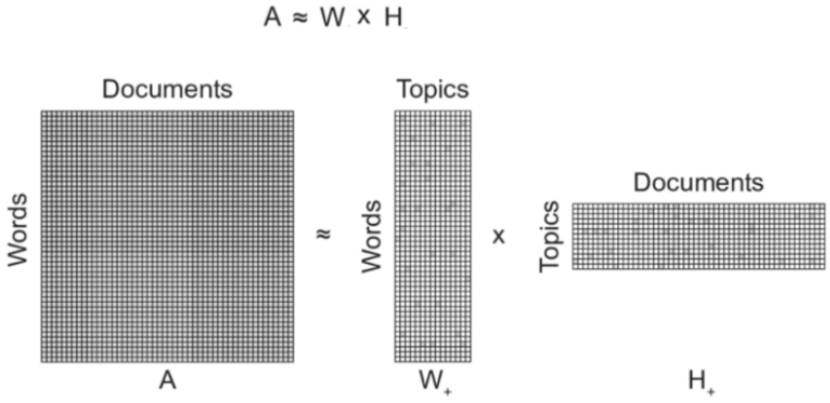
Voyons si les prédictions sont meilleures avec une autre méthode non supervisée : la méthode NMF.

## 2. Avec la méthode NMF (non-negative matrix factorization)

### a. Principe de l'algorithme

Le modèle NMF (non-negative matrix factorization) est une méthode d'algèbre linéaire qui peut être utilisée pour la prédiction de sujets. Supposons que le nombre de sujets à prédire est K. On note N le nombre de mots présents ou considérés dans le corpus et D le nombre de documents.

Alors la matrice TF-IDF de taille (N,D) que l'on note A, et qui est à termes positifs, peut se décomposer de la manière suivante :




où W et H sont eux matrices à termes positifs.

### b. Prédictions

Ensemble des thèmes avec le modèle NMF

```
In[] : display_topics(nmf, tfidf_feature_names, 15)
Out[] : Topic 0:
error user run request server test web api application net url project log work version
Topic 1:
android studio layout_width com layout_height wrap_content activity layout xml support gradle item
match_parent sdk widget
Topic 2:
file directory folder line project path open txt command upload text create bash copy include
Topic 3:
string convert object json public stre format str character number return char contain method set
Topic 4:
git branch commit master push repository github merge remote origin pull repo rebase local clone
Topic 5:
table sql query key select database mysql create row field primary foreign column server schema
Topic 6:
class public static method type extend object href span bootstrap btn override constructor col model
Topic 7:
```

```
function return var console datum javascript log jquery scope react test component object method event
Topic 8:
date day datetime month format year time hour convert number timestamp return datepicker iso end
Topic 9:
python line module print import package install pip lib usr instal script run version command
Topic 10:
image background color css text width button html height style size tag page center bootstrap
Topic 11:
array numpy object arr php convert index return byte remove store test javascript create loop
Topic 12:
list item loop collection index contain convert object arraylist type return map generic foreach dict
Topic 13:
value select option type key input property default variable set text label return dictionary number
Topic 14:
int std struct amp cout main char include return type static byte pointer const compiler
Topic 15:
java org lang method eclipse androidruntime exception util apache main activitythread springframework bean
maven jdk
Topic 16:
column dataframe panda datum row frame index csv col data series nan count sum grid
Topic 17:
app io xcode component react play device google angular store import route simulator iphone router
```

Les thèmes semblent plus précis que ceux définis avec la méthode LDA. Testons sur les mêmes exemples que précédemment:

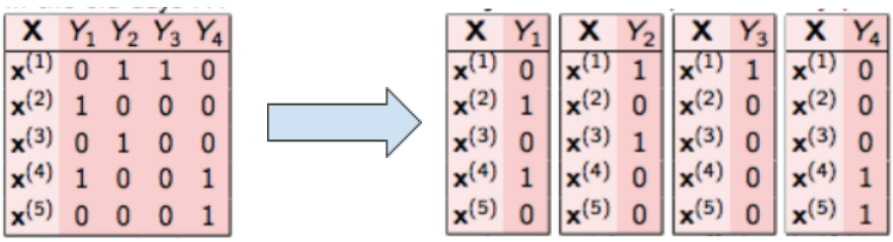
```
In [] : tags_nmf(text1)
Out [] : ['import', 'write', 'file', 'path', 'directory', 'open', 'txt', 'filename', 'create']


In [] : tags_nmf(text2)
Out [] : ['python']

In [] : tags_nmf(text3)
Out [] : ['compile', 'gcc', 'main', 'printf', 'int', 'include']
```

# III. Classification multi-labels

## 1. Méthode

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 :



## 2. Test des différents modèles

On définit une variable new_tags où on a enlevé les tags qui n'apparaissent que dans 1% des posts. Nous nous baserons sur cette variables pour les étiquettes de la classification.

```
In [] : # Preprocess text data
X=df.Preprocessed_text.values.tolist()
y = df.new_tags

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.2)

# Use tf-idf
vectorizer = TfidfVectorizer(min_df = 0.005, max_df=0.90,sublinear_tf=True)
vectorised_train_documents = vectorizer.fit_transform(X_train).toarray()
vectorised_test_documents = vectorizer.transform(X_test).toarray()

#Use multilabelbinarizer on targets
mlb = MultiLabelBinarizer()
train_labels = mlb.fit_transform(y_train)
test_labels = mlb.transform(y_test)
```

On entraîne les modèles KNN, Naive bayes et linear SVC, les deux derniers étant entraînés avec la BinaryRelavance.  Nous obtenons les scores suivants sur le jeu de test:

| | macro_f1 | micro_f1 | accuracy |
|---|---|---|---|
| **KNN** | 0.490787 | 0.557951 | 0.353955 |
| **Naive Bayes** | 0.116853 | 0.294155 | 0.160243 |
| **Linear SVC** | 0.608758 | 0.689837 | 0.451318 |

Le modèle SVC linéaire est le meilleur des trois. Nous prenons ce modèle en ajoutant une ACP dans le pre-processing du texte. Celle-ci donne des résultats à peine moins bons mais rend la prédiction plus rapide.
Voici les prédictions obtenues sur les trois exemples.

Exemple 1 :

```
In [] : predict_multi_labels2(text1)
Out [] : [('python',)]
```

Exemple 2 :

```
In [] : predict_multi_labels2(text2)
Out [] : [('python',)]
```

Exemple 3:

```
In [] : predict_multi_labels2(text3)
Out [] : [('c',)]
```

# IV. Choix du modèle et déploiement de l'app

## 1. Choix du modèle

Dans le fichier test.xlsx, nous pouvons voir les résultats des modèles pour une cinquantaine de posts. Le modèle supervisé SVC linéaire (avec ACP) est le plus pertinent. Son implémentation est développée dans /flaskapp/model.py

## 2. Création de l'application flask

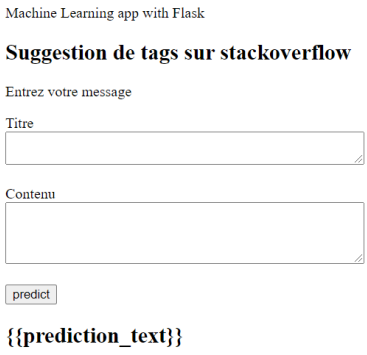
Nous détaillons ici simplement le schéma d'utilisation de l'application.
Dans le dossier flaskapp,

- *QueryResults.csv* est la base de données des posts.

- *functions_model.py* contient les fonctions qui permettent de nettoyer le texte : clean_text, remove_stop_words et lemmatize.

- *model.py* contient l'implémentation du modèle sélectionné grâce à la base de données. Il génère les fichiers *pca.pkl*, *tfidf.pk*l et *tags_model.pkl*

- *home.html* dans le dossier *templates* permet de mettre en forme l'interface utilisateur




- app.py est l'application flask. Elle prend en input le message écrit par l'utilisateur sur l'interface puis nettoie le texte du message, applique le pre-processing (tfidf et pca) et le modèle (tags_model) pour réaliser et afficher la prédiction.

Son exécution donne une adresse locale.

## 3. Déploiement sur heroku

Le déploiement en site internet s'est faite via heroku. Cela a nécessité le fichier *requirements.tx*t qui correspond à l'ensemble des packages nécessaires pour faire tourner l'application et du fichier *Procfile*. Nous avons créé l'application en passant par un repo github.

# Bibliographie


Regular expressions
https://www.regular-expressions.info/quickstart.html
https://python.sdv.univ-paris-diderot.fr/16_expressions_regulieres/?utm_source=pocket_mylist

Modèles non supervisés
https://thinkinfi.com/guide-to-build-best-lda-model-using-gensim-python/?utm_source=pocket_mylist
https://www.analyticsvidhya.com/blog/2021/07/topic-modelling-with-lda-a-hands-on-introduction/?utm_source=pocket_mylist
http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/?utm_source=pocket_mylist
https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0
https://towardsdatascience.com/topic-modeling-quora-questions-with-lda-nmf-aff8dce5e1dd

https://medium.com/ml2vec/topic-modeling-is-an-unsupervised-learning-approach-to-clustering-documents-to-discover-topics-fdfbf30e27df

Modèles supervisés
https://medium.com/technovators/machine-learning-based-multi-label-text-classification-9a0e17f88bb4
https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/?utm_source=pocket_mylist

Application flask et déploiement
https://towardsdatascience.com/develop-a-nlp-model-in-python-deploy-it-with-flask-step-by-step-744f3bdd7776
https://getpocket.com/read/2742098612


# Github Repository

https://github.com/CelineMendola/suggestion-tags-stackoverflow