

# CATEGORISEZ AUTOMATIQUEMENT DES QUESTIONS

# Introduction

Le but de ce projet est d'étudier comment modéliser et traiter des documents textuels. Plus précisément, il est demandé ici d'extraire une certaine quantité de questions sur un site web informatique assez connu, stack overflow, et de traiter ces questions afin d'en extraire des tags associés à chacune de ces questions. 3 méthodes distinctes sont demandées : non supervisées, semi-supervisé, et full supervisé.

Il est demandé ensuite, à l'aide de l'étape précédente, de créer un outil à mettre en ligne qui puisse traiter n'importe quelle question utilisateur (pas forcément issue du site de stack overflow), et d'afficher en sortie une liste de tags associée.

Le projet va donc être décomposé en plusieurs étapes :

1. Importer les données, les nettoyer et les traiter de manière à les rendre exploitables par des modèles de traitement de données
2. Créer une première approche de création de tags non supervisée, c'est-à-dire sans aucune idée à priori des tags à générer.
3. Créer ensuite une approche semi supervisée, qui va se baser sur les résultats de l'approche précédente, en y appliquant en plus une couche de classification
4. Créer enfin une approche supervisée, qui se base uniquement sur les tags utilisateurs pré renseignés pour chaque question par les auteurs des questions eux-mêmes, et y appliquer un modèle classificateur.
5. Une fois sélectionné un modèle de donnée particulier pour chacune des approches, la prochaine étape est d'adapter les étapes précédentes à n'importe quelle question, et récupérer en sortie une liste de tags associée.
6. Mettre en ligne le traitement précédent via une interface web proposant au visiteur de rentrer une question, choisir un modèle et d'afficher en sortie la liste de tags générée.

# Sommaire

Introduction .....	2
I. Import, analyse exploratoire, nettoyage et transformation des données (feature extraction) ..	4
I.1. Import des données (feature extraction) .....	4
I.2. Feature extraction .....	5
I.3. Vectorisation / Transformation en matrices tf et tf idf .....	7
II. Approche non supervisée .....	8
II.1. LDA : choix de l'algorithme et scoring .....	9
II.2. NMF : choix de l'algorithme et scoring .....	13
II.3. Tags de sortie .....	14
III. Approche semi et full supervisée .....	15
IV. Transformation et traitement d'une nouvelle question utilisateur .....	18
V. Création d'une API et d'un end point de générateur de tags .....	20
VI. Conclusion .....	22

# I. Import, analyse exploratoire, nettoyage et transformation des données (feature extraction)

## I.1. IMPORT DES DONNEES (FEATURE EXTRACTION)

Le site d'import de question Stack Overflow est accessible ici : [Query Stack Overflow - Stack Exchange Data Explorer](#)

On y rentre une requête au format SQL avec les restrictions suivantes (visant à avoir une question valide) :

- La question doit se terminer par un '?'
- Elle doit être de type PostTypeId 1 (correspond au type 'question' dans la nomenclature Stack Overflow)
- Elle doit comporter au moins 3 commentaires ou réponses et être au moins mis en favori par 5 autres utilisateurs
- On souhaite obtenir autour de 20000 questions, on va donc introduire une limitation temporelle sur la date de création de la question
- On souhaite avoir 2 champs : le champ 'Title' qui contient la question au format chaîne de caractères, et le champ 'Tags', qui contient les tags renseignés directement par l'auteur de la question.

Ceci nous donne la requête suivante :

```
SELECT Id, Title, Tags FROM Posts WHERE  
RIGHT(Title,1) = '?' AND PostTypeId = 1  
AND (CommentCount >3 OR AnswerCount>3)  
AND FavoriteCount>=5  
AND CreationDate >= '2014/01/01'
```

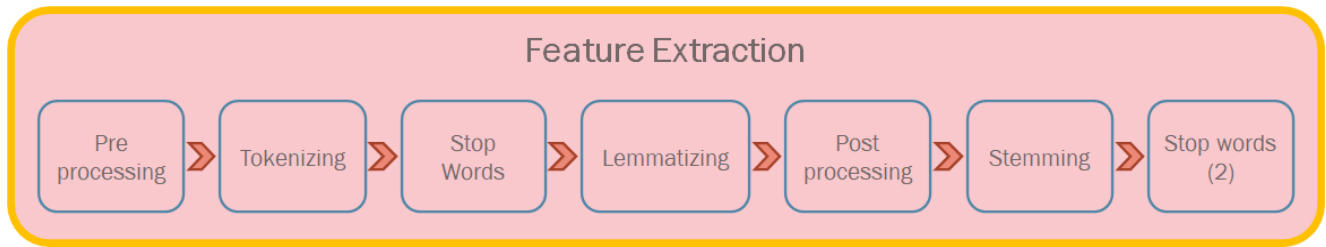
On obtient donc par la suite un dataset, une fois importé, de 2 colonnes, 'Title', et 'Tags', et un index 'id'. Une fois Title et Tags renommé de manière plus explicite, on obtient ce dataset (extrait) :

	Question	Tags_Stack_Overflow
Id		
20864872	How to bind parameters to a raw DB query in Laravel that's used on a model?	<php><mysql><pdo><laravel><laravel-4>
20864934	OPTION (RECOMPILE) is Always Faster; Why?	<sql><sql-server><sql-server-2008><compilation><hint>
20866996	How to Compress Slot Calls When Using Queued Connection in Qt?	<c++><qt><qthread><QtCore><Qt-Signals>
20869993	Why does ArrayList implement RandomAccess Interface?	<java><collections>

de taille : `df_data.shape`  
(21648, 2) , soit 21648 lignes x 2 colonnes.

## I.2. FEATURE EXTRACTION

On va chercher ici à transformer chaque question en une liste de mots représentatifs.



On va donc effectuer les opérations suivantes sur chaque élément de notre colonne de questions :

**Preprocessing** : on remplace les ' \_ ' par des '-'

**Tokenization** : Au moyen d'une expression régulière à déterminer, on va transformer notre question en une première liste de mots.

```
RegExp = r'[\da-zA-Z\+\#\&]+'
```

L'expression régulière choisie transforme en mot tout ce qui comporte des chiffres, lettres de l'alphabet classiques minuscules et majuscules de a à z, et certains caractères spéciaux +, # &.

**Stop words** : On utilise ensuite le corpus nltk pour supprimer une liste de mots trop utilisés dans la langue anglaise et donc sans valeur significative : *nltk.stopwords*

**Lemmatization** : On utilise la librairie WordNet du corpus nltk via la fonction *nltk.stem.WordNetLemmatizer()* pour effectuer cette transformation.

**Post processing** : cette phase est composée de plusieurs fonctions visant à nettoyer les mots issus de la phase précédente.

- On enlève certains caractères en début ou fin de mot
- On supprime les chiffres en représentation hexadécimale
- On conserve certaines lettres (C, D, R, O) qui, en informatique, ont une signification spécifique (C et R sont des langages de programmation par exemple)
- On supprime les mots ne contenant pas de lettres.

**Stemming** : phase optionnelle, puisqu'elle retourne une racine de mots, et non un mot en lui-même. Si on effectue cette phase, on va avoir en sortie des tags « racinisés » (et il faudra donc une fonction inverse, forcément avec pertes, de stemming inverse)

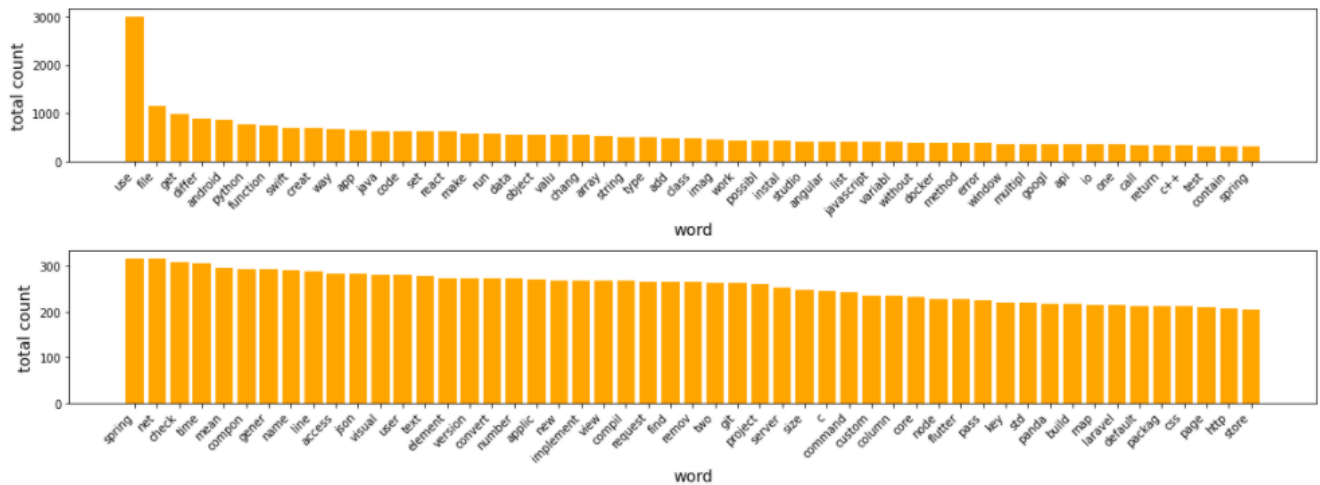
Une fois cette phase effectuée, certaines questions vont se retrouver en sortie avec des listes vides, sans mot (ce qui signifie que la question posée est trop générique et/ou ne comporte pas de mots valides).

On supprime ces questions, et on passe à **21642** questions (8 ont été supprimées).

On obtient une 3eme colonne 'Qu\_tokenized', représentant chaque question sous forme de liste :

	Question	Tags_Stack_Overflow	Qu_tokenized
Id			
30543605	How to add footer to NavigationView - Android support design library?	[android, android-support-library, android-design-library]	[add, footer, navigationview, android, support, design, librari]

L'ensemble des mots issus des listes de 'qu\_tokenized' va représenter le vocabulaire de départ, dont on peut regarder la distribution :



**Stop words (2) :** On voit sur les schémas précédents que certains mots sont sur-utilisés (use, file, get...) ou trop communs. On va les supprimer manuellement des listes de tags en les rajoutant manuellement à la liste des stop words :

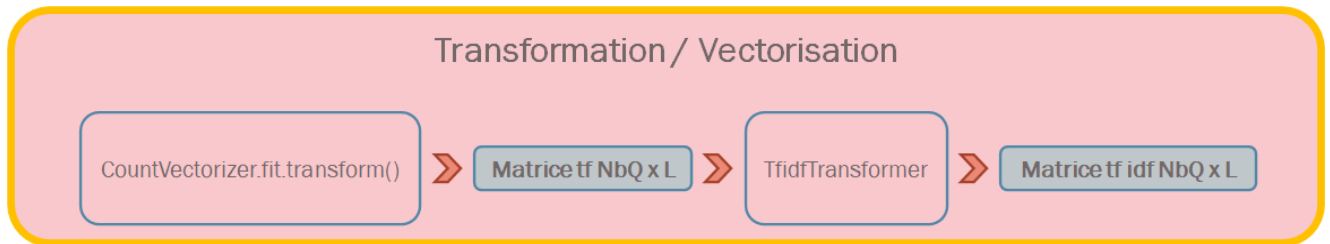
```
['use', 'get', 'way', 'differ', 'creat', 'valu', 'make', 'data', 'chang', 'best', 'work', 'run', 'possibl', 'variabl', 'without', 'one', 'find', 'check', 'line', 'name', 'number', 'text', 'multipl', 'call', 'convert', 'element', 'implement', 'return', 'two', 'user', 'mean', 'remov', 'page', 'good', 'project', 'view', 'write', 'new', 'like', 'size', 'column', 'control', 'default']
```

On resupprime de nouveau les questions dont la liste est vide de mots suite à l'étape précédente, et on obtient au final **21634** questions :

	Question	Tags_Stack_Overflow	Qu_tokenized
21629	How can I deprecate a C++ header?	[c++, deprecated]	[deprec, c++, header]
21630	What happens when mandatory RVO is applied to a reference that's extending the lifetime of a tem...	[c++, language-lawyer, return-value-optimization]	[happen, mandatori, rvo, appli, refer, extend, lifetim, temporari]
21631	How to make a background app open a dialog box every day at 2pm with Flutter?	[android, ios, flutter, dart, alarm]	[background, app, open, dialog, box, everi, day, pm, flutter]
21632	How can I distinguish between high- and low-performance cores/threads in C++?	[c++, multithreading, performance, intel, apple-m1]	[distinguish, high, low, perform, core, thread, c++]
21633	Why are static methods allowed inside a non-static inner class in Java 16?	[java, inner-classes, java-16]	[static, method, allow, insid, non, static, inner, class, java]

Les mots uniques issus de la colonne 'Qu\_tokenized' représente le vocabulaire de départ (racinisé si stemming) du corpus de questions.

### I.3. VECTORISATION / TRANSFORMATION EN MATRICES TF ET TF IDF



Une fois que notre colonne 'Qu\_tokenized' est prête sous forme de liste de mots, la dernière étape consiste à transformer cette colonne sous forme matricielle, avec une information sur la fréquence des mots dans la question et dans l'ensemble de documents.

2 représentations fréquentielles sont possibles :

- Matrices tf (on compte les mots dans chaque question)
- Matrice tf idf (on compte les mots, mais on pondère de manière inversée le résultat par la représentation du mot dans les autres questions).

Pour effectuer cette transformation, on va utiliser la librairie `sklearn.feature_extraction.text` et les fonctions `CountVectorizer` & `TfidfTransformer`

A noter que `CountVectorizer` comporte dans ses paramètres des fonctions de pre-processing, tokenization, lemming, stemming, etc. On peut donc directement inclure les étapes du § 1.2 dans le `CountVectorizer`, et obtenir notre matrice tf en sortie :

```
tf = CVect.fit_transform(df_data.Question)
tf
```

```
<21634x8539 sparse matrix of type '<class 'numpy.int64''>'
  with 102660 stored elements in Compressed Sparse Row format>
```

Notre matrice `tf-idf` s'obtient à partir de la matrice `tf` et de `TfidfTransformer` :

```
transformer = TfidfTransformer(smooth_idf=False)
tfidf = transformer.fit_transform(tf) #sparse matrix array
```

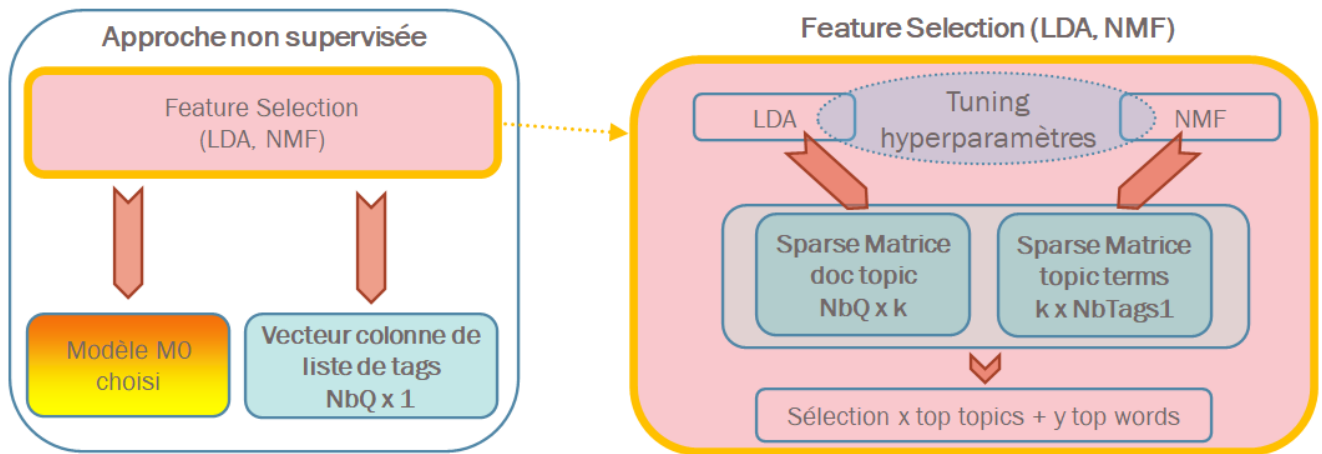
```
tfidf
```

```
<21634x8539 sparse matrix of type '<class 'numpy.float64''>'
  with 102660 stored elements in Compressed Sparse Row format>
```

Format des matrices : 21634 x 8539

- Autant de lignes que de questions : 21634
- Autant de colonnes que de mots distincts (1 champ = 1 mot issu du vocabulaire de départ) : 8539
- Les champs sont accessibles directement via `CountVectorizer.get_feature_names()`

## II. Approche non supervisée



Ici, nous partons en entrée de matrices tf ou tf idf avec de 8539 champs (features ou labels).

Le but va être :

- D'utiliser un algorithme de type 'génératif probabiliste' spécifiquement adapté au traitement de documents, afin de classifier les questions en sujet ('topics') et les 'topics' en mots représentatifs, qui seront nos tags
  - o L'algorithme retenu est celui de **Latent Dirichlet Allocation (LDA)**
- Ou de procéder à une réduction de dimension de features appliquée au traitement de texte, de façon à avoir en sortie une liste de mots réduite (dimension moindre), qui seront nos tags associés aux questions de départ.
  - o L'algorithme de **NMF (non negative matrix factorization)** va être utilisé pour cela : il va décomposer la matrice d'entrée (tf-idf) en 2 matrices plus 'simples', le but sera alors de minimiser l'erreur inhérente au process.

En sortie, nous aurons :

- Un vecteur colonne de taille NbQ (le nombre de questions), dont chaque élément sera une liste de tags.
- Le(s) modèle(s) nmf et ou lda retenus, à sauvegarder pour l'utilisation postérieure du modèle pour de nouvelles questions

Il s'agit bien d'une approche non supervisée, puisque nous n'avons aucune connaissance des tags à priori.



## II.1. LDA : CHOIX DE L'ALGORITHME ET SCORING

### Principe

L'algorithme LDA effectue du 'topic modeling', il est utilisé pour classer automatiquement des documents en un nombre  $k$  de sujets. Le LDA permet également d'extraire, à partir des sujets, une liste de mots relatifs, auxquels un poids est associé. Plus le poids est élevé, plus le mot est représentatif du sujet.

**Le paramètre d'entrée de l'algorithme LDA est la matrice  $tf$ .**

**Le nombre de sujets est, pour le LDA, un paramètre de départ à fixer.**

**LDA retourne en sortie :**

- Une matrice doc/topics :
  - o On l'obtient concrètement via la librairie sklearn, et la fonction `lda.transform()` permet de transformer  $tf$  en une matrice documents/topics, de dimension :  $NbQ \times k$ .
  - o Chaque colonne représente un sujet particulier, chaque ligne représente une question. Chaque valeur dans cette colonne représente une probabilité que le document (la question) soit représenté par ce sujet.
- Une matrice topic/terms, de dimension  $k \times L$ , où  $L$  représente le nombre de mots possibles, c.à.d. la taille du vocabulaire de départ obtenu après tokenization de l'ensemble des questions ( $L=8539$  dans notre cas orécis).
  - o On l'obtient via la fonction `lda.components_`
  - o Chaque champ est un mot possible, chaque ligne est un sujet, chaque valeur est le poids du mot associé dans le topic
  - o La liste des champs (le vocabulaire de départ) est déjà connue, c'est la même que celle de la matrice  $tf$  d'entrée.

**Hyperparamètres à faire varier :**

- Le nombre de sujets/topics  $k$
- Le paramètre alpha (entre 0 et 1) : paramètre de Dirichlet de départ (à priori), pour la distribution document / topic
- Le paramètre beta ( $>0$ ) : paramètre de Dirichlet de départ pour la distribution topic / terms

Concrètement, voici les différentes valeurs testées ci-dessous :

```
n_topics_list = [5, 10, 30, 50, 70, 100] #number of topics
alpha_1       = np.arange(0.01, 1, 0.3)   # Document topic density
beta_1        = [0.01, 0.5, 1, 10]        # Word topic density
```

D'autres hyperparamètres auraient également pu être modifiés :

- learning\_decay (rythme d'apprentissage)-
- learning\_offset (paramètre permettant de réduire l'importance des premiers apprentissages)
- max\_iter (nombre maximum d'itérations).

Cela n'a pas été le cas pour des raisons de limitations de puissance du PC de travail (et donc du temps de traitement).

**Evaluations et sélection des modèles :**

- **Scoring** : 3 méthodes de scoring ont été appliquées.
  - o **Score de perplexité** : `lda.perplexity(tf)`, qui doit être le plus bas possible
  - o **Log likelihood** : `lda.score(tf)`, qui doit être le plus haut possible. Ce score est directement lié au score de perplexité
  - o **Coherence score** : Méthode issue de la librairie gensim dont on peut trouver plus de détails ici : <https://radimrehurek.com/gensim/models/coherencemodel.html>

Voir ce [paragraphe](#) pour la représentation graphique des scores selon les hyperparamètres choisis.

- **Visualisation graphique des distributions de topics et de mots**
  - o On vérifie, pour le ou les modèles retenus par scoring, que les répartitions document/topics, et topic/terms sont relativement homogènes.
  - o En effet, si un topic est ultra prépondérant quelle que soit la question, on obtiendra au final toujours les mêmes tags. De même, si, indépendamment du topic, ce sont toujours les mêmes mots qui ont les poids les plus élevés, on obtiendra également toujours les mêmes tags au final.

Voir ce [paragraphe](#) pour la représentation graphique des distributions de topics et de mots par question/topics.

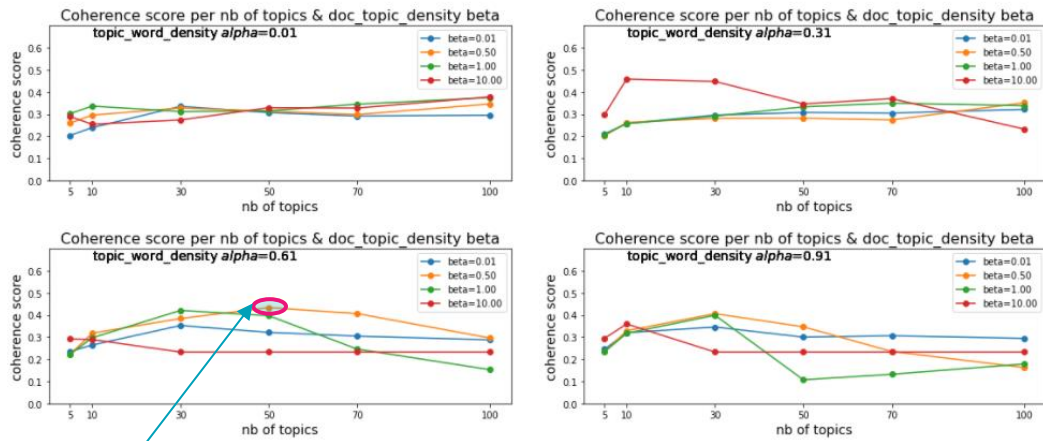
### Modèle lda retenu :

- Un paramètre **beta** élevé ne donnant pas de bons résultats (un topic est ultra prioritaire sur tous les autres)
- **beta** à 0.01 donne de très mauvais scores de perplexité et log likelihood
- => on a fixé **beta** à 0.5
- On a trié par score de cohérence
- Résultat : k = 50 topics, alpha = 0.61, beta = 0.5

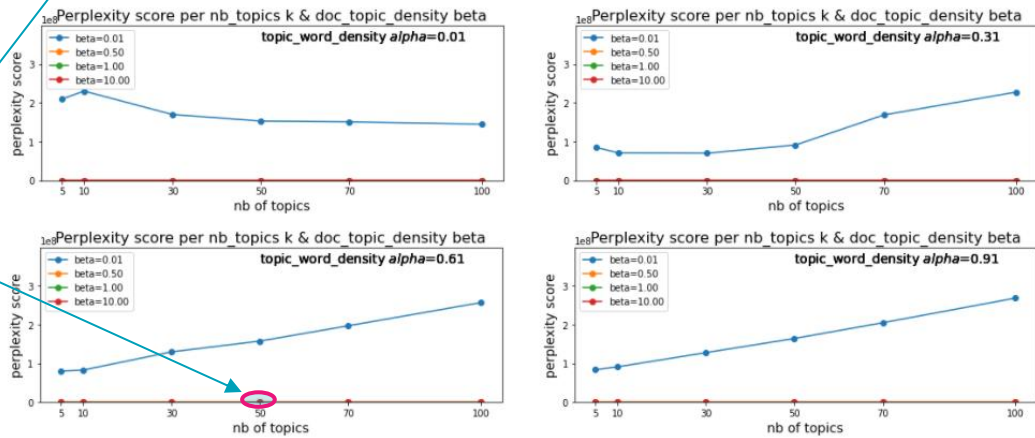
	n_topics	alpha	beta	coh_score	perp_score	log_lik_score
57	50	0.61	0.5	0.433111	8721.422527	-9.603795e+05
73	70	0.61	0.5	0.406296	12527.323778	-9.987088e+05
45	30	0.91	0.5	0.406117	6130.424489	-9.230676e+05
41	30	0.61	0.5	0.384084	5865.700008	-9.183954e+05
85	100	0.31	0.5	0.351860	25668.977443	-1.074638e+06

## Représentation graphique des 3 scores (tuning des hyperparamètres du modèle LDA)

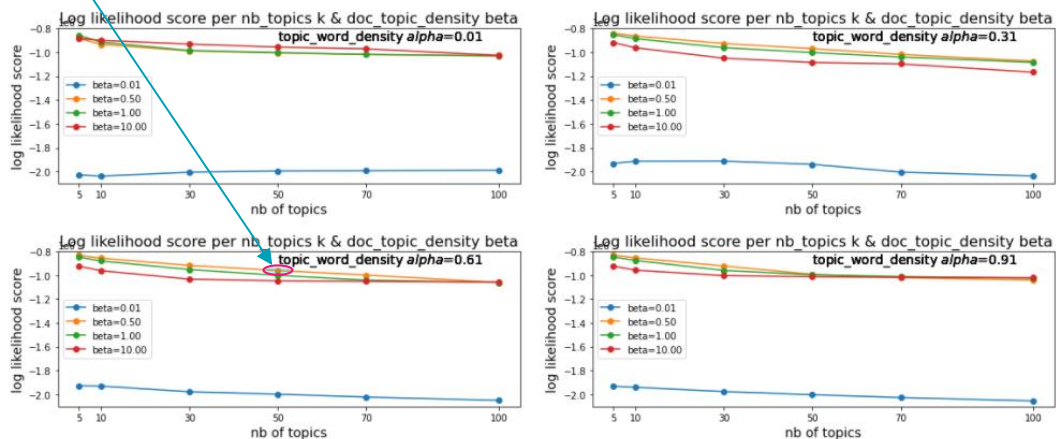
### Coherence score



### Perplexity score



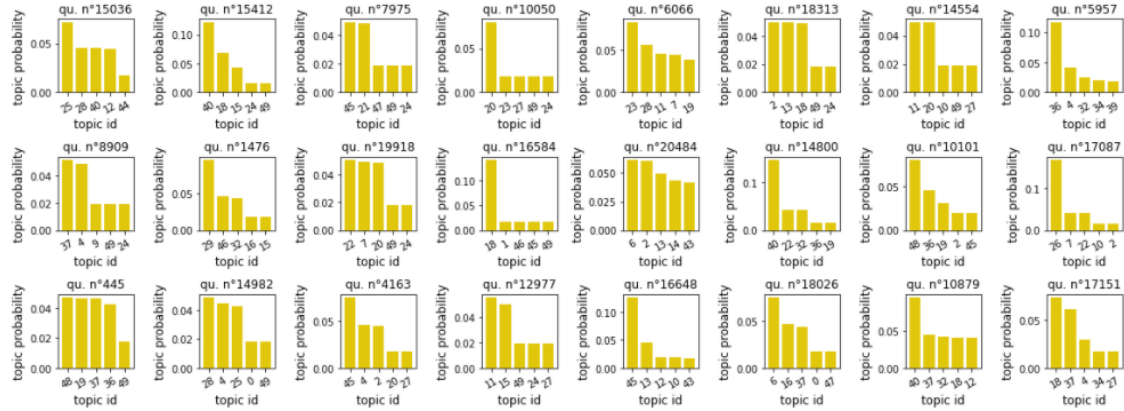
### Log likelihood score



Modèle retenu

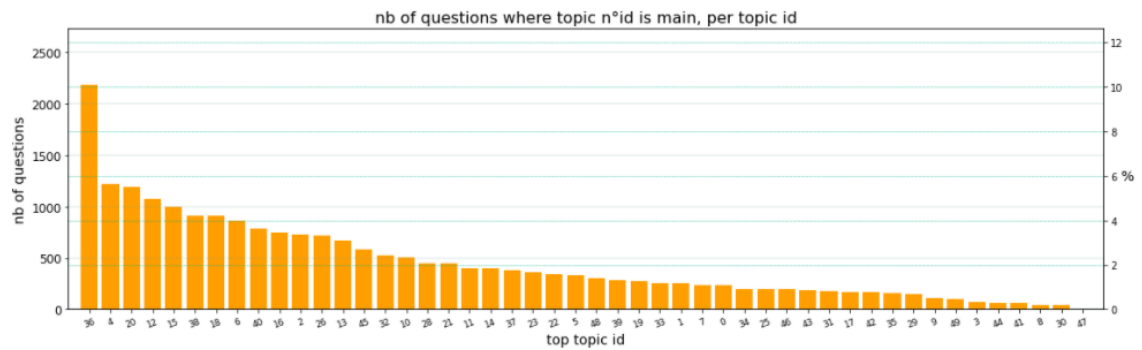
## Visualisation graphique des distributions de topics et de mots pour le modèle lda retenu

```
: display_topx_topics_per_question(df_doc_topics_lda01, ncolx = 8, TopT = 5, question_min=0, nb_questions = 24, random = True)
```

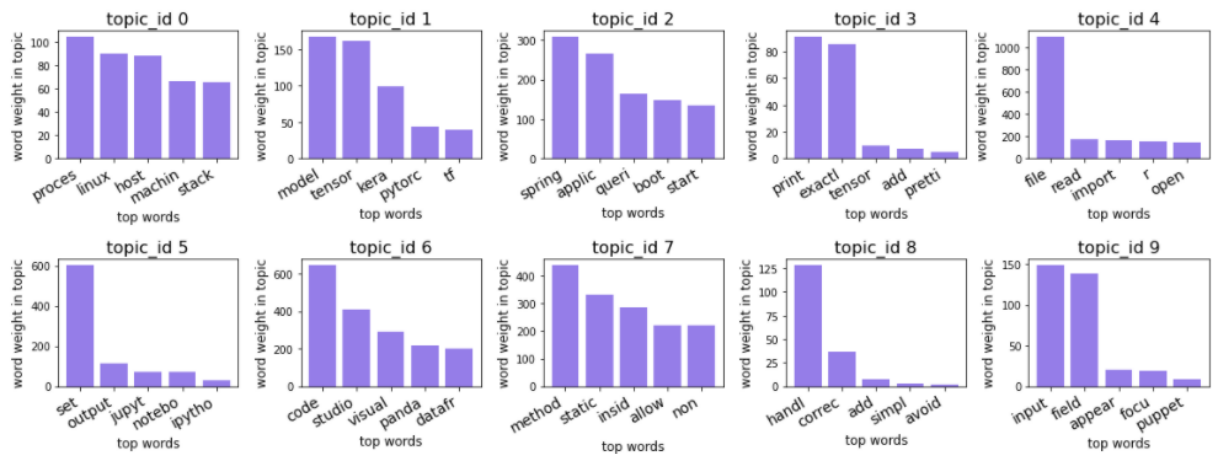


Représentation des 5 Topics principaux pour 24 questions prises au hasard

```
display_top_topic_count(df_doc_topics_lda01, nbars = 60)
```



Distribution des topics : nombre de questions qu'ils représentent en tant que topic principal



Représentation des 5 mots principaux pour les 10 premiers topics

## II.2. NMF : CHOIX DE L'ALGORITHME ET SCORING

### Principe

NMF (Factorisation de matrice non négative) est un algorithme de réduction de dimension et de décomposition matricielle, utilisable uniquement dans le cas de matrice avec éléments positifs.

Il consiste à décomposer une matrice de départ (notre matrice *tf-idf* dans notre cas), en un produit de matrices plus 'simple'.

La différence entre le produit et la matrice de départ représente l'erreur, que l'on va chercher à minimiser.

Concrètement, on cherchera à décomposer notre matrice comme suit :

$$tfidf = WH + \varepsilon$$

Il s'agira alors de minimiser la norme de l'erreur  $\|\varepsilon\|_?$  de  $\varepsilon$ , où le type de norme '?' est un des paramètres à choisir.

Si l'on veut en plus ajouter des paramètres de régularisation  $\alpha$  et  $L1\_ratio$  ( $=\rho$ ) (similaire à la régularisation 'elasticNet' en régression linéaire), la fonction de perte à minimiser devient :

$$\frac{\|tfidf - WH\|_?^2}{2} + \alpha \times \left( \rho(\|W\|_1 + \|H\|_1) + \frac{(1-\rho)(\|W\|_{fro}^2 + \|H\|_{fro}^2)}{2} \right)$$

Où  $\| \cdot \|_1$  est la norme 1 ( $\sum | \cdot |$ ) et  $\| \cdot \|_{fro}^2$  la norme-2 ou norme de Froebenius ( $\sqrt{\sum | \cdot |^2}$ )

L'algorithme NMF retourne en sortie les matrices W et H, accessibles respectivement via `nmf.transform()` et `nmf.components_`. Pour notre cas, W va représenter la matrice document/topics, de taille NbQ x k, et H la matrice topic/terms de taille k x L, avec :

- NbQ = nombre de questions
- k = nombre de sujets
- L = taille du vocabulaire de départ. Les L champs de H sont ceux de *tf idf*.

### Hyperparamètres à faire varier :

- Nombre de topics k
- Paramètres  $\alpha$  et  $\rho$  de régularisation
- Beta-loss : Type de norme à appliquer pour le calcul de  $\|tfidf - WH\|$
- Solver : algorithme de résolution utilisé pour minimiser la fonction de perte ('cd' ou 'mu')

Dans la pratique, les valeurs suivantes ont été testées :

```
n_topics_list = [5, 10, 30, 50, 70, 100]
alpha_list = [0, 0.1, 0.5, 1, 2, 5, 10]
l1_ratio_list = np.arange(0, 1.1, 0.2)
beta_loss_list = ['frobenius', 'kullback-leibler']
solver_list = ['cd', 'mu']
```

### Evaluations et sélection des modèles :

Ici, l'unique méthode de scoring est la norme de la fonction d'erreur à minimiser, que l'on récupère via `nmf.reconstruction_err_`. On fait varier les différents paramètres comme indiqué plus haut et on récupère l'entrée correspondant au minimum de `nmf.reconstruction_err_`:

```
k = 100
norme = 'frobenius'
solver = 'cd' (coordinates descent)
alpha = 0 (= pas de régularisation)
```

```
#we look for the min of nmf_err
nmf_err_min = d_hyperparam.nmf_err.min()
ind_min = d_hyperparam.query("nmf_err == " + str(nmf_err_min)).index[0]

nmf1 = nmf[ind_min] #best algorithm
d_hyperparam.loc[ind_min,:]

n_topics      100
bl            frobenius
solver         cd
alpha          0
l1r
nmf_err       131.075
Name: 555, dtype: object
```

## II.3. TAGS DE SORTIE

Nos algorithmes lda et nmf ayant été déterminés, on va pouvoir déterminer les tags associés à chaque question.

### Principe :

Pour chaque question, on va récupérer dans la matrice doc topics ( $W$ ) les  $x$  sujets principaux (ceux dont la probabilité est la plus forte), et pour ces  $x$  sujets, les  $y$  mots principaux (ceux avec le plus de poids).

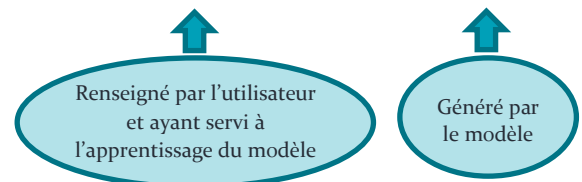
**On va donc récupérer en tout  $x \times y$  mots par question, qui seront nos tags de sortie.**

Fixons  $x = 1$  et  $y = 3$  pour nmf et lda, ce qui nous donnera 3 tags générés en non supervisé par lda et nmf respectivement pour chaque question.

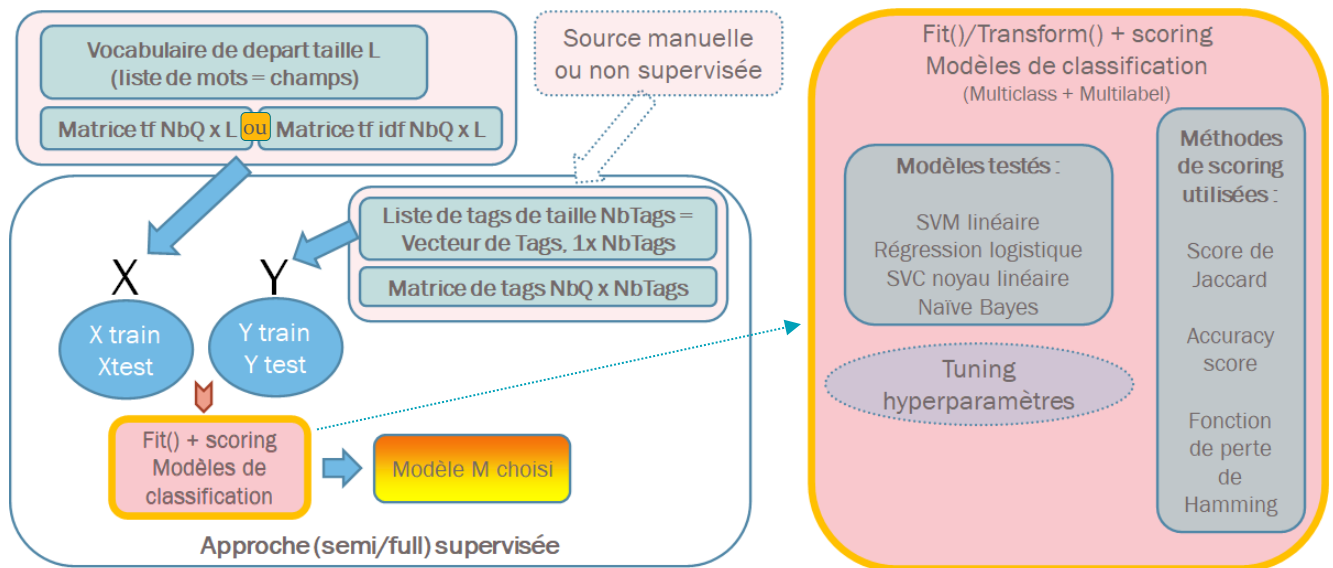
Si on a appliqué une opération de stemming en amont sur les questions, alors les tags en sortie vont être racinisés, et il faut leur appliquer une fonction de **racinisation inverse**, qui consiste à récupérer un mot utile parmi l'ensemble représentant la racine du tag.

### Résultat (extrait) pour le nmf non supervisé :

	Question	Qu_tokenized	Tags_Stack_Overflow	tags_nmf_unsupervised
0	How to bind parameters to a raw DB query in Laravel that's used on a model?	[bind, paramet, raw, db, queri, laravel, model]	[php, mysql, pdo, laravel, laravel-4]	[parameter, queri, url]
1	OPTION (RECOMPILE) is Always Faster; Why?	[option, recompile, alway, faster]	[sql, sql-server, sql-server-2008, compilation, hint]	[select, option, group]
2	How to Compress Slot Calls When Using Queued Connection in Qt?	[compress, slot, queue, connect, qt]	[c++, qt, qthread, qtcore, qt-signals]	[server, sql, connect]
3	Why does ArrayList implement RandomAccess Interface?	[arraylist, randomaccess, interfac]	[java, collections]	[method, static, interface]
4	How to add Activity to manifest.xml in right way?	[add, activ, manifest, xml, right]	[android, android-activity, android-manifest]	[add, attribute, conditon]



### III. Approche semi et full supervisée



17

#### Principe :

Les approches semi et full supervisées utilisent des modèles de classification pour générer les tags. Il s'agit donc de les entraîner et les tester sur nos input/output (question tokenisée /tags), et de retenir le modèle avec le meilleur score.

On parle de (semi) supervisé car les tags (nos labels de classification) sont déjà pré-existants au modèle :

- Dans le cas full supervisé, les tags utilisés sont ceux renseignés par l'utilisateur (2<sup>ème</sup> colonne du dataset généré au tout début : 'Tags Stack Overflow').
- Dans le cas semi supervisé, les tags utilisés sont ceux qui ont été créés par l'approche non supervisée (colonne 'tags\_lda' ou 'tags\_nmf' de sortie de l'algorithme retenu)

#### Données d'entrée et de sortie des modèles :

##### Génération des input X :

Approche supervisée : les données sont celles de la matrice tf

Approche semi-supervisée : on utilise les données de la matrice tf dans le cas de tags issus de l'algorithme lda, ou tfidf pour des tags issus de l'algorithme nmf.

Les champs sont les L mots issus du vocabulaire de départ.

##### Génération des output Y :

On part du vecteur colonne de tags, que l'on va transformer en dataset dummy (0 ou 1). Les champs seront la liste de tags uniques, les lignes seront les questions.

On utilise `MultilabelBinarizer()` pour générer cette matrice : on le 'fit' au vecteur colonne de liste de tags, et on transforme.

##### Datasets d'entraînement et de tests :

On découpe de manière classique X et Y en dataset d'entraînement et de tests (80% / 20%) à l'aide de `train_test_split()`.

## Types et Modèles de classification testés :

On se trouve ici dans le cas d'une classification multiclasse (autant de classes que de tags possibles) et multi label (on peut avoir, pour une question donnée, plusieurs labels associés, cad plusieurs tags)

### Modèles testés :

- Linear SVM (support vector machine) avec SGD (Descente de Gradient stochastique)
- Logistic Classifier (régression logistique adaptée à la classification)
- Linear SVC (support vector classifier avec noyau linéaire)
- Classificateur de Naive Bayes polynomial

### Méthodes de scoring utilisées :

- 'accuracy' score (très peu permissif et peu adapté au multi label)
- 'jaccard' score : adapté à la classification multilabel
- 'hamming loss' score : plus il est bas, mieux c'est

Une fois le modèle fitté aux données d'entraînement  $X_{train}$  &  $Y_{train}$ , on va effectuer une prédiction de  $X_{test}$ , que l'on va comparer à l'output réel  $Y_{test}$  via les méthodes de scoring.

### Résultats et tuning :

Concrètement, afin de pouvoir effectuer du multiclasse multilabel, on va utiliser la fonction **OneVsRestClassifier** sur chaque modèle de scikitlearn.

	SGDClassifier Linear SVM with gradient descent minimum loss	LogisticRegression Logistic Classifier	LinearSVC Support Vector Classifier linear Kernel	MultinomialNB Naïve Bayes
LDA tags (semi sup.)	0 / 0 / 0,03	0,43 / 0,45 / 0,02	0,47 / 0,54 / 0,02	0,22 / 0,23 / 0,036
NMF tags (semi sup.)	0 / 0 / 0,01	0,57 / 0,64 / 4,9e-3	0,59 / 0,68 / 4,7e-3	0,16 / 0,19 / 0,01
St. Ov. Tags (full sup.)	0 / 0 / 5,9e-4	0,11 / 0,30 / 4,6e-4	0,14 / 0,36 / 4,7e-4	0,03 / 0,19 / 1,1e-3

Accuracy score / Jaccard Score / Hamming

Pour l'approche semi supervisée, l'approche NMF donne de meilleurs résultats que LDA.

Par ailleurs, c'est le modèle **SVC linéaire** qui donne les meilleurs résultats pour les 2 approches semi et full supervisées.

On va donc faire varier les paramètres du SVC linéaire pour l'optimiser, en utilisant en plus de la validation croisée avec **GridSearchCV** (k splits).



```
k=5

parameters = {'estimator__C': [0.1, 0.5, 1, 10],
               'estimator__loss': ('hinge', 'squared_hinge'),
               'estimator__dual': [False, True],
               'estimator__penalty': ('l1', 'l2'),
               'estimator__fit_intercept': [False, True]}
```

### Résultat final :

Approche semi supervisée via algorithme NMF et modèle SVC Linéaire optimisé :

```
{'estimator__C': 1,
 'estimator__dual': False,
 'estimator__fit_intercept': False,
 'estimator__loss': 'squared_hinge',
 'estimator__penalty': 'l1'}
```

Approche full supervisée via modèle SVC linéaire optimisé :

```
{'estimator__C': 1,
 'estimator__dual': False,
 'estimator__fit_intercept': True,
 'estimator__loss': 'squared_hinge',
 'estimator__penalty': 'l1'}
```



## Récupération des tags de sortie :

En sortie du modèle, on obtient une matrice dummy Y avec une liste de champs (les tags) et des 1 quand le tag est retenu pour la ligne (la question) concernée.

- ⇒ On applique une fonction inverse qui transforme la matrice dummy en un vecteur colonne contenant une liste de tags par ligne

Exemple pour les tags issus de l'approche supervisée avec modèle LSVC, à comparer aux tags renseignés par l'utilisateur :

```
df_data.sample(50)[['Question', 'Tags_Stack_Overflow', 'tags_supervised_stovf']]
```

	Question	Tags_Stack_Overflow	tags_supervised_stovf
1902	Android: How to programatically login to website and retrieve data from it?	[java, android, post, httpclient, httpURLConnection]	[android]
8391	Is there an elegant way to split a file by chapter using ffmpeg?	[ffmpeg]	[ffmpeg]
11991	what is 'z' flag in docker container's volumes-from option?	[docker]	[docker]
21334	Why is this "Hello, World!" JavaScript code fragment recognized as an acceptable program instruc...	[javascript, node.js]	[android-fragments, javascript]
7972	What is the difference between addListener(event, listener) and on(event, listener) method in no...	[node.js]	[javascript, node.js]
15610	Is there a function for generating settings.SECRET_KEY in django?	[django, django-settings]	[django, python]
13115	How to iterate object keys using *ngFor?	[angular, typescript, object, ngfor]	[angular]
14485	How to get/set Trello custom fields using the API?	[api, trello]	[api, trello]
14298	How can I use/create dynamic template to compile dynamic Component with Angular 2.0?	[angular, typescript, compilation, angular2-compiler]	[angular, typescript]

Renseigné par l'utilisateur  
et ayant servi à  
l'apprentissage du modèle

Généré par  
le modèle

Exemple une fois tous les modèles générés et appliqués aux questions de départ :

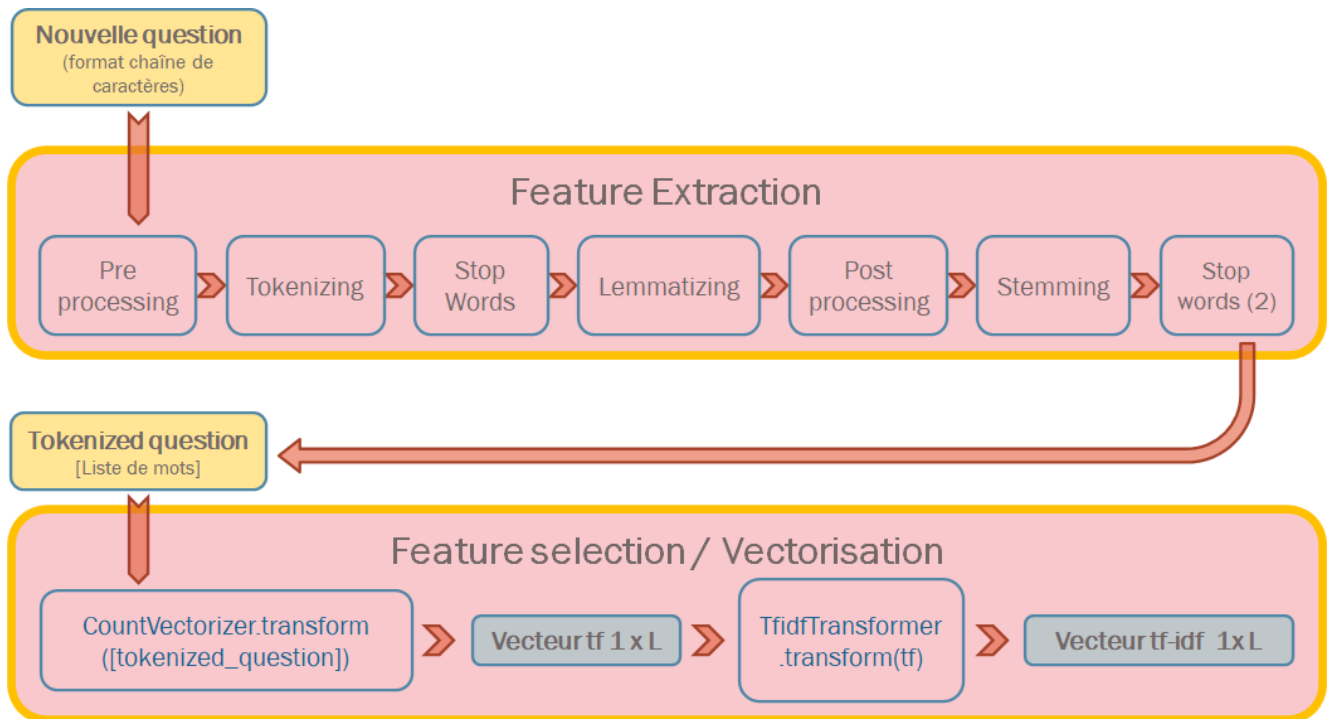
	Question	Tags_Stack_Overflow	tags_lda_unsupervised	tags_nmf_unsupervised	tags_semi-supervised_nmf	tags_supervised_stovf
11023	Global npm install location on windows?	[node.js, windows, npm, pycharm]	[install, window, laravel]	[window, console, bash]	[bash, console, window]	[node.js]
15050	How to specify (optional) default props with TypeScript for stateless, functional React components?	[javascript, reactjs, typescript, react-native]	[react, component, access]	[select, option, group]	[group, option, select]	[javascript, reactjs]
275	Aptana Error-pydev: Port not bound (found port -1)?	[python, aptana, pydev]	[error, disable, fix]	[error, fix, fail]	[error, fail, fix]	[aptana, python]
4034	Value cannot be null. Parameter name: value, CreateIdentityAsync?	[c#, asp.net-mvc, customization, asp.net-identity]	[list, html, template]	[parameter, queri, url]	[parameter, queri, url]	[asp.net-mvc, c#]
15076	Angular2-router: How to only change a parameter of a route?	[angular, angular2-routing]	[react, component, access]	[parameter, queri, url]	[parameter, queri, url]	[angular]
10262	Swift startsWith method?	[ios, swift]	[method, static, inside]	[method, static, interface]	[interface, method, static]	[swift]
9978	Wordpress: Get the select field option choices in ACF (Advanced Post Field) plugin?	[php, wordpress, advanced-custom-fields]	[input, field, appear]	[select, option, group]	[group, option, select]	[php, wordpress]
21087	How do I configure absolute paths for imports in TypeScript based React Native apps?	[javascript, typescript, react-native, jestjs, babeljs]	[file, read, import]	[react, native, router]	[native, react, router]	[react-native, typescript]
10886	Why does this invalid-looking code compile successfully on g++ 6.0?	[c++, g++]	[compile, css, load]	[compile, program, c#]	[c#, compile, program]	[c++, g++]
16895	How to iterate through a nested dict?	[python, dictionary]	[object, update, properti]	[loop, event, iter]	[event, iter, loop]	[python]

## IV. Transformation et traitement d'une nouvelle question utilisateur

### Transformation de la question en input tf ou tfidf

La transformation d'une nouvelle question utilisateur suit la même démarche qu'au §I. Le but est de transformer la question utilisateur en un vecteur ligne tf ou tfidf, qui servira d'input pour l'un des 3 modèles (non supervisé / semi supervisé / full supervisé).

On utilise pour cela **les mêmes fonctions** `CountVectorizer` et `TfidfTransformer` qu'au §I., **déjà fittées**, qui effectueront une transformation de la nouvelle question :



### Génération des tags en sortie

La génération des tags en sortie se passe exactement de la même manière que pour les étapes précédentes : on réutilise les mêmes modèles déjà 'tunés' et fittés, pour chacune des approches, et on applique la fonction `predict()` ou `transform()`.

#### Approche non supervisée :

- Pour les algorithmes nmf et lda, la matrice topic-terms H (`lda` ou `nmf.components_`) ne change pas suivant la question et dépend uniquement du modèle fitté.
- Le vecteur W question-topics est calculé via `lda.transform(tf)` ou `nmf.transform(tfidf)`
- On récupère de la même manière qu'avant  $x$  topics principaux et  $y$  mots principaux associés aux  $x$  topics, ce qui nous donne  $x \times y$  tags en sortie.

#### Approche semi-supervisée ou full supervisée :

- On applique directement au modèle LSVC que l'on a 'tuné' et 'fitté' précédemment :
  - o la fonction `predict(tf)` dans le cas full supervisé et semi-supervisé avec lda
  - o `predict(tfidf)` dans le cas semi supervisé avec nmf

- On obtient en sortie un vecteur 'dummy' composé de 0 ou de 1 dont on extrait les tags retenus.

Dans la pratique, on a créé une **fonction** qui effectue directement toutes les étapes précédentes et qui prend en entrée :

- la question utilisateur sous forme de chaîne caractères
- le modèle voulu (soit l'algorithme lda ou nmf de l'approche non supervisée, soit un des modèle LinearSVC des approches semi et full supervisées)
- un booléen indiquant si l'approche est supervisée ou non
- la liste de mot du vocabulaire de départ (après stemming)
- la liste de tags qui sont les champs du vecteur ligne de sortie du modèle de classification pour les approches semi et full supervisés uniquement
- la liste de mots du vocabulaire de départ avant stemming (Vocab), pour effectuer le stemming inverse des tags retenus dans le cas d'une approche non supervisée

Cette fonction traite également les erreurs (vocabulaire inconnu ou trop générique et donc bloqué par les stopwords).



```
##### ----- Create_tags final Function ----- #####
def create_tags(input_user, model, words_list_for_training, tag_list, Vocab = [], unsupervised = False):
```

## V. Création d'une API et d'un end point de générateur de tags

### Principe et architecture

#### Côté serveur :

- L'API est créée sur un hébergeur web gratuit Heroku à l'aide du framework Flask, et avec le serveur web gunicorn.
- La configuration du serveur se fait via 2 fichiers principaux:
  - o *'Requirements.txt'*, qui indique à Heroku l'environnement nécessaire à un bon fonctionnement du programme et les librairies à installer
  - o *'Procfile'*, qui indique la configuration des process et quel script lancer au démarrage :
    - 1<sup>ère</sup> ligne : `web: gunicorn init:app`
    - Ici on lui indique de lancer le serveur web gunicorn, et de lancer l'application Flask *'app'* située dans le fichier *'init.py'*
- Il est important de lancer la commande en ligne suivante, pour activer le ou les process web sur le serveur :
  - o `heroku ps:scale web=1`

#### Côté client :

- La page web consiste à demander à l'utilisateur de :
  - o Renseigner une question
  - o Sélectionner un des 3 modèles proposés (non supervisé avec LDA, semi supervisé avec NMF et LSVC, ou supervisé avec LSVC)
  - o Cliquer pour lancer la recherche.
- Une seule recherche est possible (le bouton disparaît dès que la recherche est lancée).
- Le lancement de la recherche va permettre l'émission d'une requête POST au serveur web, qui va effectuer son traitement via une tâche de fond (cf § suivant)

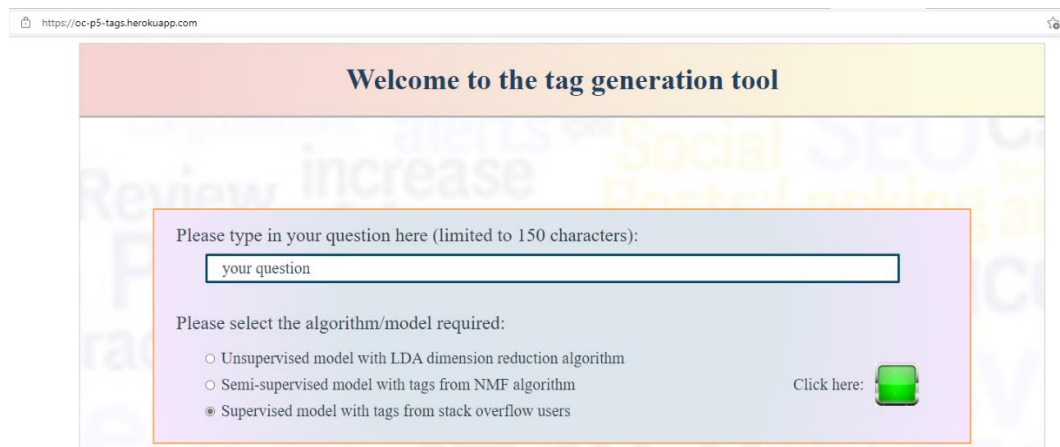
#### Temps de traitement et tâches de fond côté serveur :

- Le temps de traitement d'une page web sur Heroku étant limité à un maximum de 30s, il a été décidé, afin de contourner cette limitation, de lancer la génération de tags en 'background', via des jobs, ou tâches de fond.
- On utilise pour cela Redis et RQ, adaptés pour l'hébergeur Heroku.
  - o RQ permet de lancer des 'jobs' en tâche de fond, sans que le serveur web doive attendre une réponse pour continuer.
  - o Les tâches de fond, par définition, ne sont pas soumis au timeout de Heroku. On peut par contre configurer un timeout (10 minutes, 1heure ...) sur la ou les tâches de RQ, timeout au delà duquel la tâche est passée en échec (status *'failed'*)
- Le fichier *'Procfile'* doit allouer au moins un process *'worker'* aux traitements des tâches de fond, en lui indiquant quel script jouer au démarrage
  - o 2<sup>ème</sup> ligne : `worker: python worker.py`
  - o Ici on lui indique qu'un process worker va être utilisé et qu'il doit exécuter le script *'worker.py'* dès qu'il est lancé
    - *'worker.py'* indique au process d'établir une connexion au serveur Redis et lui dit de se mettre en attente d'une tâche à exécuter.
- Il est important de lancer une autre commande en ligne pour démarrer le process worker en question, via la commande en ligne suivante : `heroku ps:scale worker=1`

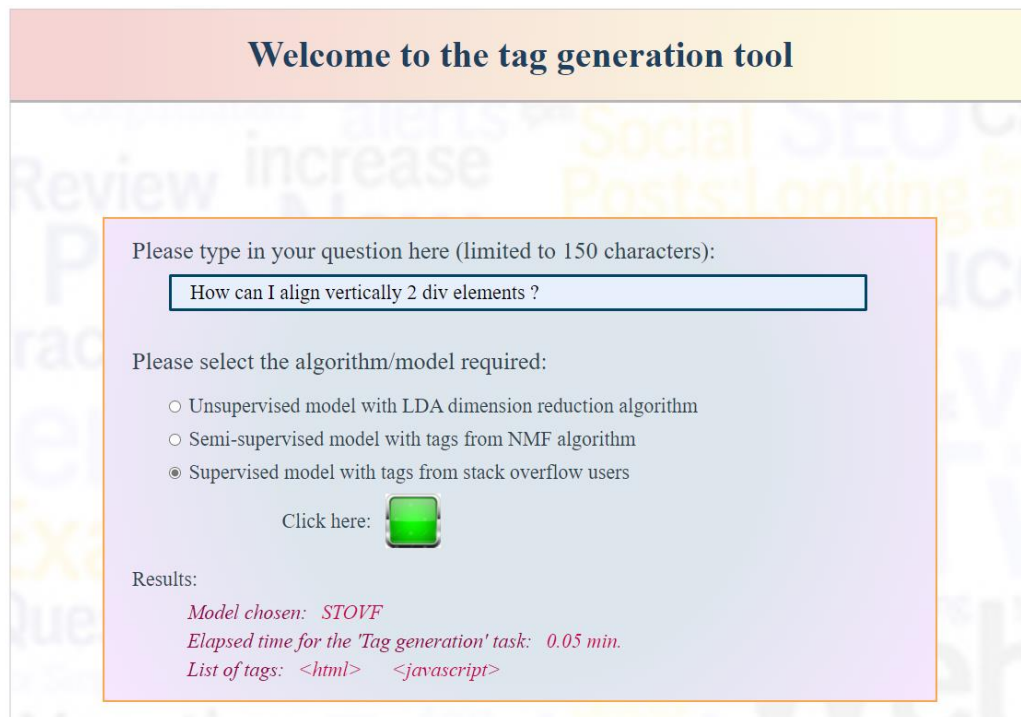
### Tâches de fond côté client :

- Une fois la tâche de fond lancée côté web, il s'agit alors, côté navigateur web client :
  - o De récupérer l'ID de la tâche
  - o De lancer via jquery et ajax un script qui va effectuer une requête au serveur web toutes les secondes pour vérifier si/quand la tâche est terminée
  - o Dès que l'information est remontée comme quoi la tâche est terminée, le navigateur affiche les résultats sur la page web client

### Ecran d'accueil :



### Ecran une fois les tags générés :



## VI. Conclusion

**Les résultats donnés par les différentes approches sont de qualité différente :**

- L'approche full supervisée semble donner les meilleurs résultats, même si le nombre de tags est parfois restreint (1 ou 2 tags souvent, rarement plus)
- L'approche nmf semi supervisée semble donner de bons résultats et est un bon compromis quand on ne dispose pas d'un ensemble de classes (les tags) de départ prédéfinis.
- L'approche lda non supervisée ne donne pas des tags forcément pertinents => il est sans doute nécessaire d'approfondir le réglage des hyperparamètres du modèle
- 

**Pistes d'amélioration :**

- Afin de traiter correctement cette demande de génération de tags, on peut utiliser un dataset beaucoup plus grande avec des machines beaucoup plus puissantes
- La phase exploratoire de feature selection peut largement être améliorée sur plusieurs aspects :
  - o Ne pas se limiter aux représentations fréquentielles (tf ou tf-idf) mais aussi aux approches plus complexes prenant en compte les spécificités liées au langage (bigramme, trigrammes, n grammes...)
  - o On peut également tester des bibliothèques de stemming et lemmatisation plus adaptées au langage informatique, et plus riches que ce qui est proposé de base dans nltk
- Les approches non supervisées sont améliorables sur plusieurs points :
  - o Plutôt que de retenir arbitrairement x topics et y top words par question, on peut très bien réfléchir à un seuil à dépasser pour le(s) topic(s) retenu(s) dans la matrice doc/topic, et pour le(s) mot(s) retenu(s) dans la matrice topic/terms
  - o Outre lda et nmf, d'autres approches non supervisées de feature selection/topic modeling pourraient également être testées
- Concernant les approches semi/full supervisées, d'autres modèles pourraient être utilisés, notamment des algorithmes de réseaux de neurones, ou des modèles de clustering type K-means.