

Catégorisez automatiquement des questions

L'objectif de ce projet est de développer un système de suggestion de tags pour une question posée sur le site Stack Overflow. Le but est d'aider les membres du site à mieux classifier leurs questions et avoir des réponses potentiellement plus pertinentes.

Dans un premier temps, je vais récupérer les données à partir d'une API du site Stack Overflow, puis je vais les analyser et traiter en utilisant des méthodes propres au traitement du langage naturel afin d'en tirer tout leur potentiel.

Dans un second temps, je vais mettre en œuvre 2 approches différentes de recommandation de tags. La première, non supervisée, visera à trouver le sujet principal d'une question et à proposer des mots relatifs au sujet détecté. La seconde, supervisée, visera à généraliser, à des questions non classifiées, les tags des questions déjà classifiées fournis par l'API Stack Overflow.

Le système de recommandation de tags mettant en œuvre les 2 approches sera intégré au travers d'une simple application web. Pour finir, des ouvertures à l'amélioration seront proposées.

Table des matières

Récupération des données.....	1
Analyse de la variable <i>Tags</i>	2
Natural Language Processing de la variable <i>Body</i>	4
Préprocessing des données avant modélisation	5
Transformation de la variable <i>Tags</i>	5
Séparation des jeux de données en jeux d'entraînement et validation	5
Transformation de la variable <i>Body</i> en « bag of words ».....	6
Analyse non supervisée.....	6
Méthode Latent Dirichlet Allocation	6
Analyse supervisée	9
Indice de Jaccard	9
Binary Relevance	9
Transformation de la variable <i>Body</i> avec TermFrequency-InverseDocumentFrequency	9
Dummy Classifier	10
Recherche par grille du meilleur modèle	10
MultinomialNB	10
LogisticRegression	10
API	11
Pistes d'amélioration.....	12

Récupération des données

L'API de Stack Overflow permet de requêter en SQL les diverses données publiques nécessaires au projet, notamment le contenu de la question ainsi que les tags associés.

Je ne récupère que les documents de type « Question » et leur note qui va me permettre de m'assurer de la qualité de leur contenu. J'effectue plusieurs requêtes en filtrant sur l'identifiant des documents, puis exporte le résultat dans un fichier CSV, jusqu'à obtenir un volume de documents suffisant.

Exemple de la dernière requête effectuée :

```
select Posts.Id,
       Name,
       Score,
       Body,
       Tags
from Posts
  inner join PostTypes on Posts.PostTypeId = PostTypes.id
where PostTypes.Name = 'Question'
  and Posts.Id >= 550000
  and Posts.Id < 600000
```

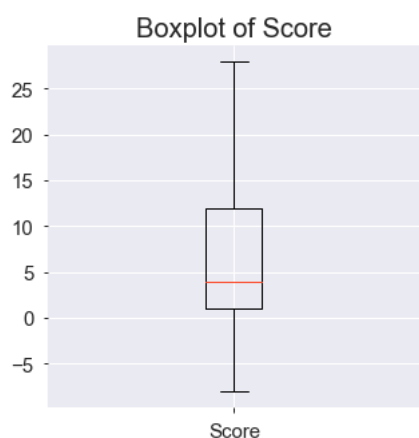
Je concatène tous les fichiers CSV dans un dataframe. J'obtiens alors un jeu de données qui comporte **91 947 questions**.

J'utilise le module Pandas Profiling afin de m'aider à explorer les données. Il en ressort notamment que le jeu de données ne possède **aucune valeur manquante et aucune question en doublon**.

Exemple d'un document :

Id	121656
Name	Question
Score	3
Body	<p>I have the following string and I would like to remove <code>&lt;bpt *&gt;*&lt;/bpt&gt;</code> and <code>&lt;ept *&gt;t*&lt;/ept&gt;</code> (notice the additional tag content inside them that also needs to be removed) without using a XML parser (overhead too large for tiny strings).</p> <pre><code>The big &lt;bpt i="1" x="1" type="bold"&gt;&lt;b&gt;&lt;/bpt&gt;black&lt;ept i="1"&gt;&lt;b&gt;&lt;/ept&gt; &lt;bpt i="2" x="2" type="ulined"&gt;&lt;u&gt;&lt;/bpt&gt;cat&lt;ept i="2"&gt;&lt;/u&gt;&lt;/ept&gt; sleeps.\r\n</code></pre><p>Any regex in VB.NET or C# will do.</p></p></pre>
Title	Regular expression to remove XML tags and their content
Tags	<code><c#><.net><xml><vb.net><regex></code>

La colonne *Body* est parsemée de sauts de ligne ou balises HTML et devra faire l'objet d'un nettoyage spécifique. Je vais la fusionner avec la colonne *Title*.

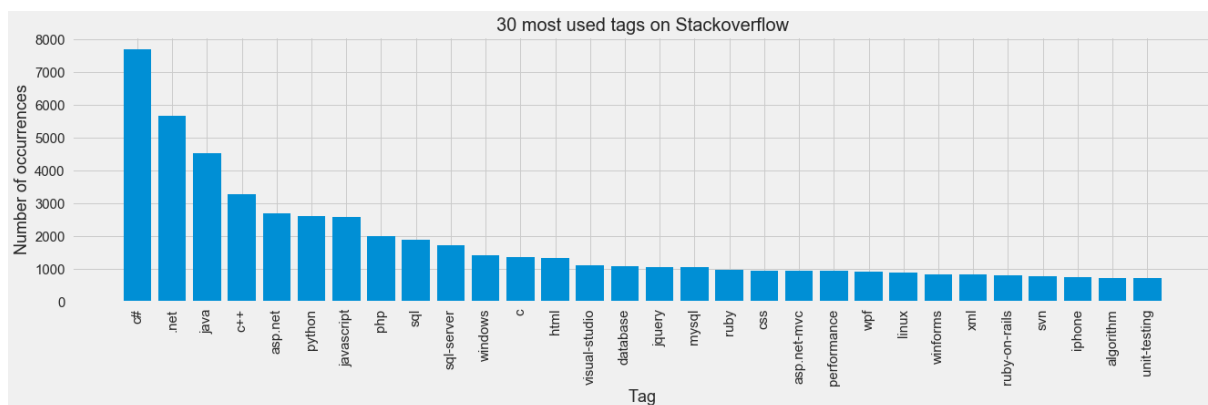


La colonne *Score* présente une distribution asymétrique très étalée à droite avec une médiane à 4.

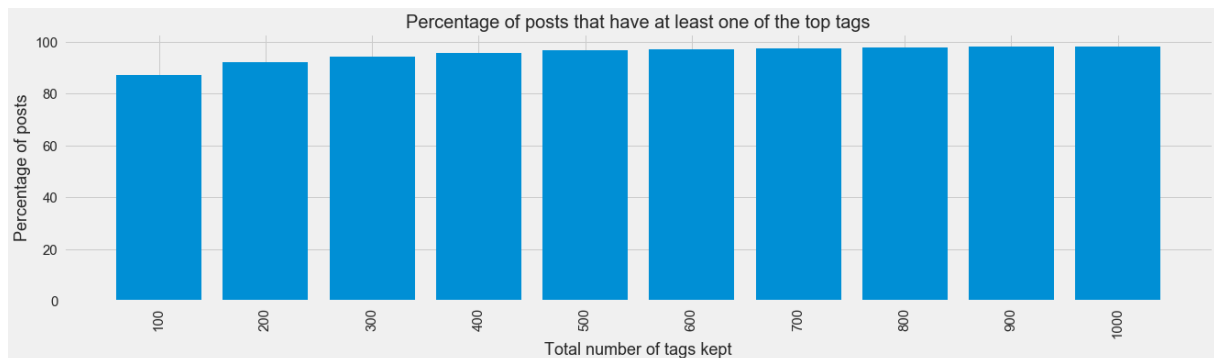
Pour m'assurer de ne traiter que des documents présentant un niveau de qualité suffisant, je ne conserve que ceux dont **le score est supérieur ou égal à 3, soit 55 598 documents**.

Analyse de la variable *Tags*

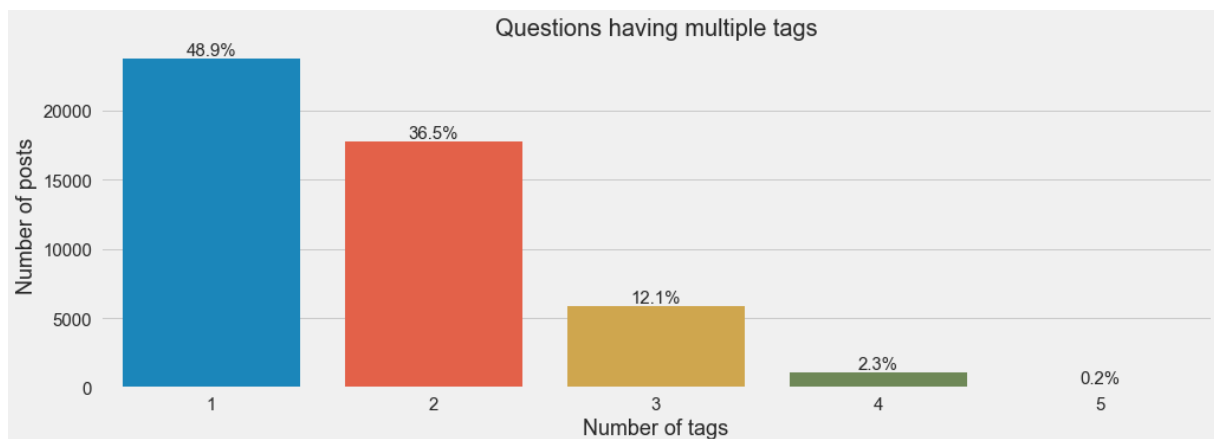
La variable *Tags* comporte **9 712 tags distincts**. Je calcule le nombre d'occurrences de chaque tag dans le dataset et produit un graphique du top 30. Le tag `<c#>` apparaît 7 703 fois et représente 9.42% des tags du dataset puis on tombe très rapidement sur des fréquences inférieures à 1% (`<unit-testing>` en fin de distribution sur le graphique apparaît 739 fois pour une fréquence de 0.9%)



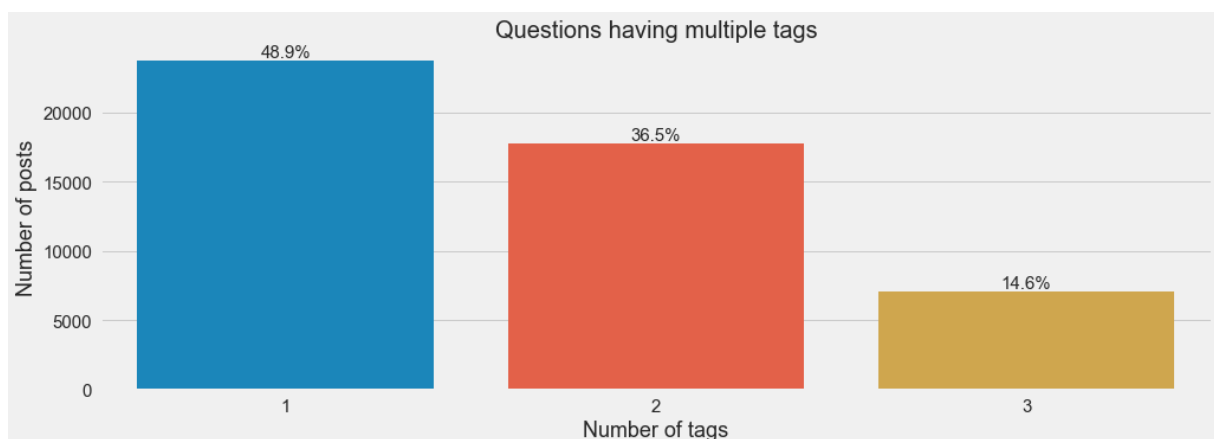
Etant donné le grand nombre de tags, je vais faire preuve de parcimonie et ne conserver qu'un nombre réduit de tags. D'une part, je m'affranchirais en partie du fléau de la dimension, cela me sera plus facile de prédire le bon tag à partir d'un nombre réduit de mots ; d'autre part, dans les presque 10 000 tags, certains sont extrêmement confidentiels et spécifiques.



Ne conserver que les 100 tags les plus fréquents me semble être un bon compromis dans la mesure où ils apparaissent au moins une fois dans 89% des tags des documents. Je vais donc supprimer les tags exclus de la colonne *tags* et supprimer les documents qui ne possèdent plus de tags. Le dataset possède désormais **48 527 documents**.



Un document peut posséder jusqu'à 5 tags. En réalité, je constate que très peu de documents (1 209) ont plus de 3 tags. Toujours pour des raisons de parcimonie et considérant que 3 tags sont suffisants pour décrire une question, **je ne conserve dans la colonne *tags* que les 3 tags les plus fréquents** ce qui m'amène à la nouvelle distribution suivante.



Natural Language Processing de la variable *Body*

Comme j'ai pu le constater précédemment, la variable *Body* nécessite d'être nettoyée et transformée pour être exploitable. Je vais donc lui appliquer des méthodes de processing propre au traitement du langage naturel :

```
40240 <p>I have the following string and I would like to remove <code>&lt;bpt *&gt;*&lt;/bpt&gt;</code> and <code>&lt;ept *&gt;*&lt;/ept&gt;</code> (notice the additional tag content inside them that also needs to be removed) without using a XML parser (overhead too large for tiny strings).</p>\r\n\r\n<pre><code>The big &lt;bpt i="1" x="1" type="bold"&gt;&lt;b&gt;&lt;/bpt&gt;black&lt;ept i="1"&gt;&lt;b&gt;&lt;/ept&gt; &lt;bpt i="2" x="2" type="ulined"&gt;&lt;u&gt;&lt;/bpt&gt;cat&lt;ept i="2"&gt;&lt;u&gt;&lt;/ept&gt; sleeps.\r\n</code>></pre>\r\n\r\n<p>Any regex in VB.NET or C# will do.</p>\r\nRegular expression to remove XML tags and their content
```

- Mise en minuscules du texte, suppression des caractères « whitespace » et du code au motif que le vocabulaire contenu entre des balises code est à mon sens trop spécifique.

```
40240 <p>i have the following string and i would like to remove and (notice the additional tag content inside them that also needs to be removed) without using a xml parser (overhead too large for tiny strings).</p><pre></pre><p>any regex in vb.net or c# will do.</p>regular expression to remove xml tags and their content
```

- Suppression du format HTML avec le package Beautiful Soup.

```
40240 i have the following string and i would like to remove and (notice the additional tag content inside them that also needs to be removed) without using a xml parser (overhead too large for tiny strings). any regex in vb.net or c# will do. regular expression to remove xml tags and their content
```

- Recodage des top tags possédant des caractères spéciaux dans les documents pour éviter des effets de bord indésirables lors de la suppression de la ponctuation ou ultérieurement, lors de la tokenisation.
- Suppression de la ponctuation.

```
40240 i have the following string and i would like to remove and notice the additional tag content inside them that also needs to be removed without using a xml parser overhead too large for tiny strings any regex in xyzspecialtags16zyx or xyzspecialtags26zyx will do regular expression to remove xml tags and their content
```

- Suppression des stopwords proposés par les modules NLP (NLTK et Spacy).
- Lemmatisation pour réduire les mots à leur forme neutre canonique.
- Suppression des mots qui ne sont pas des noms (POS tagging) considérant que les verbes ou adverbes n'apportent pas de valeur ajoutée pour la problématique.

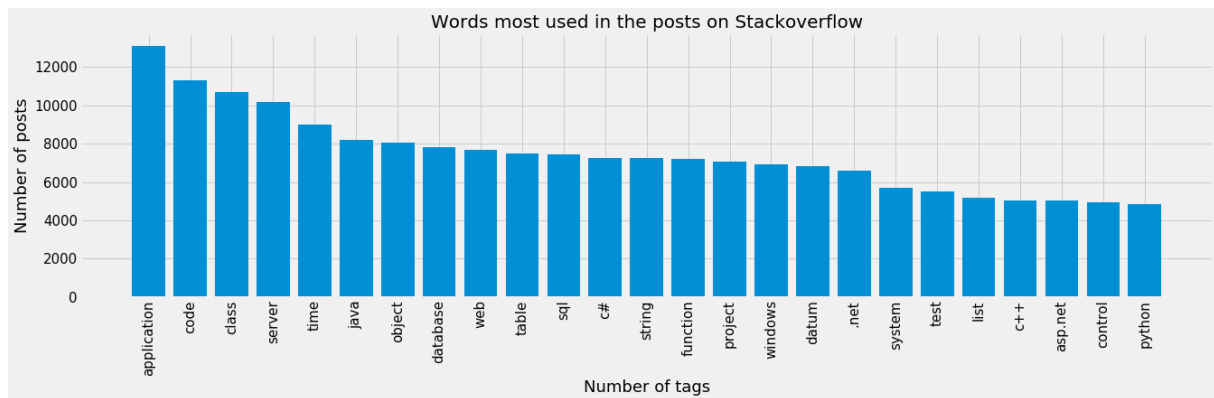
```
40240 string notice tag content xml parser string regex xyzspecialtags16zyx xyzspecialtags26zyx expression xml tag
```

- Suppression de stopwords manuels. A ce stade, un rapide coup d'œil sur les 200 premiers mots par occurrence met en évidence que certains noms sont très génériques et n'apportent pas de valeur ajoutée, ils sont supprimés.

```
'file', 'way', 'user', 'use', 'problem', 'work', 'example', 'method', 'question', 'value', 'thank', 'solution', 'thing', 'number', 'change', 'idea', 'answer', 'issue', 'update', 'lot', 'message', 'information', 'people', 'reason', 'help', 'want', 'run', 'need', 'end', 'default', 'difference', 'suggestion', 'approach', 'task', 'implementation', 'check', 'e', 'custom', 'place', 'practice', 'support', 'experience', 'product', 'stuff', 'comment', 'note', 'argument', 'year'
```

Conclusion après NLP :

- Tous les documents possèdent au moins un mot.
- Il y a **24 103 mots distincts, 914 109 en tout et environ 19 mots par document en moyenne.**
- Le top 25 des mots les plus fréquents ci-dessous met surtout en avant des mots propres aux langages informatiques :



Préprocessing des données avant modélisation

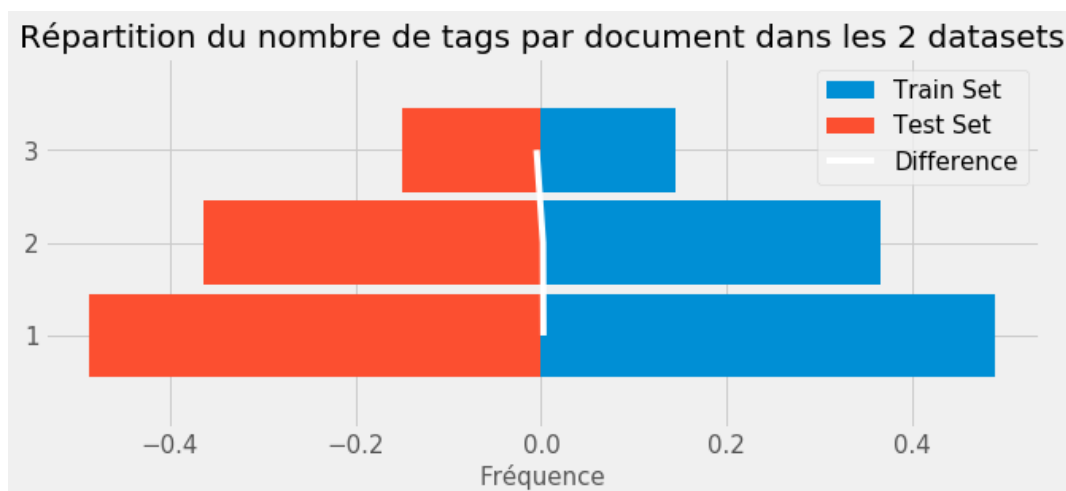
Transformation de la variable Tags

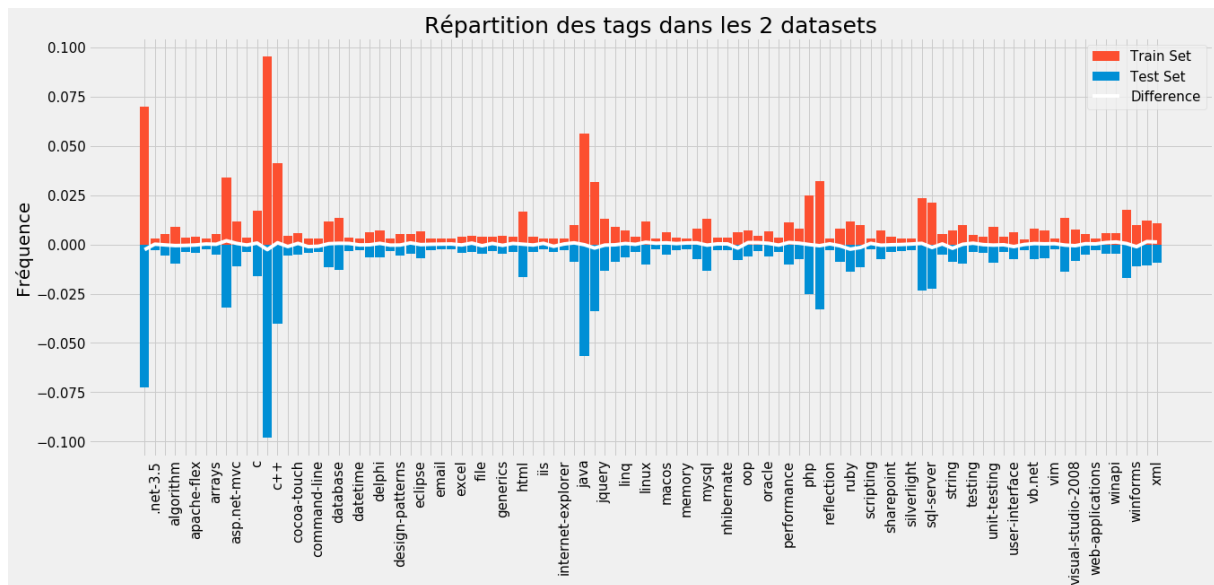
Je commence par transformer la variable *Tags* à l'aide d'un `MultiLabelBinarizer` pour la modélisation supervisée. J'obtiens une matrice de taille **(documents x tags)** soit **(48 527 x 100)** de **valeurs binaires** indiquant la présence ou non d'un ou plusieurs tags pour chaque document.

	Tag 1	Tag 2	Tag 3					
Document 1	Python			MultiLabel Binarizer				
Document 2	Pandas	Numpy	Python					
Document 3	Database	SQL						
	Python	Pandas	Numpy	Database	SQL			
Document 1	1	0	0	0	0			
Document 2	1	1	1	0	0			
Document 3	0	0	0	1	1			

Séparation des jeux de données en jeux d'entraînement et validation

Je splitte les jeux de données *Tags* et *Body* en jeux d'entraînement (80% soit **38 821 documents**) et validation (20% soit **9 706 documents**). Afin de m'assurer de l'équilibre de la répartition des tags dans les jeux d'entraînement et validation, je produis 2 pyramides, la première, de la répartition du nombre de tags par document dans les 2 datasets, et la seconde, de la fréquence d'apparition des tags dans les 2 datasets.





NB : je n'affiche qu'un tag sur 2

Je constate que la courbe représentant la différence entre les fréquences observées des tags dans le jeu d'entraînement et dans le jeu de validation reste stable autour de 0. Les jeux de données sont donc bien équilibrés.

Transformation de la variable *Body* en « bag of words »

Un bag of words est une matrice de dénombrement des mots dans le corpus qui va alimenter le LDA. Etant donné la grande dimension de mon vocabulaire, je vais chercher à le réduire en fixant une valeur pour le paramètre *min_df* de la méthode *CountVectorizer* afin de ne pas intégrer des mots trop rares. J'itère sur plusieurs valeurs entières jusqu'à obtenir des résultats qui me paraissent satisfaisants pour l'analyse non supervisée. Après plusieurs itérations, je retiens ***min_df = 150*** ce qui permet de réduire mon vocabulaire à **585 mots**, soit une matrice d'entraînement de dimension **38 821 x 585**.

Par ailleurs, je ne prends en compte que les *unigrams* et modifie le *token_pattern* par défaut pour tenir compte des mots d'un seul caractère (par exemple, le langage C).

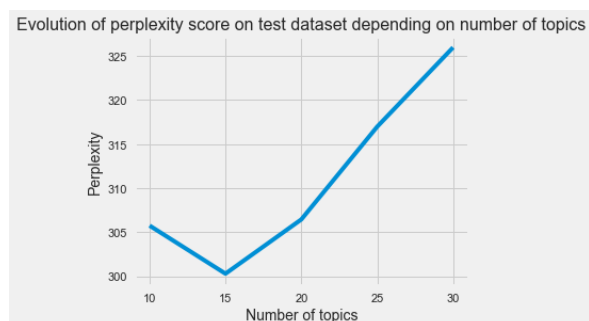
Analyse non supervisée

Méthode Latent Dirichlet Allocation

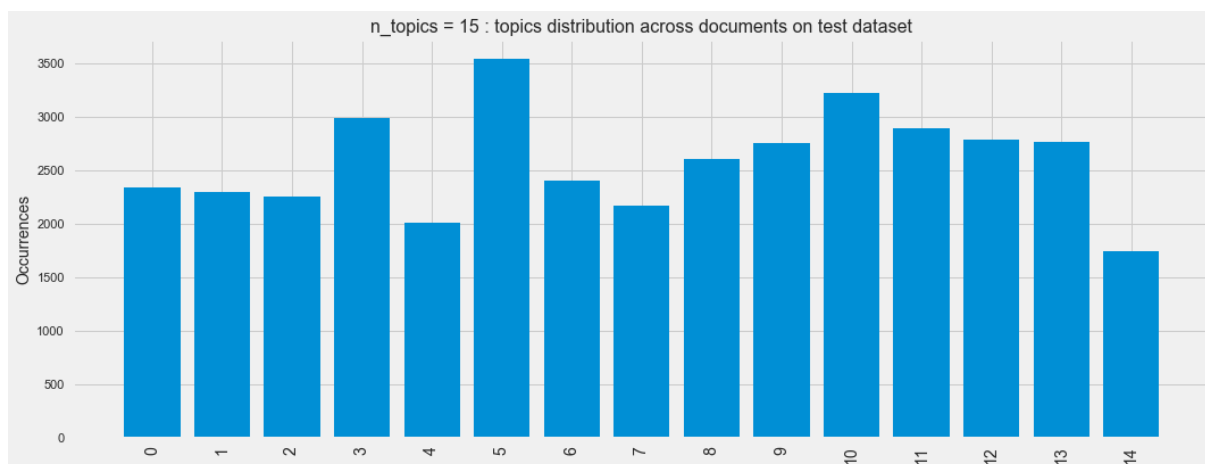
LDA suppose que chaque document d'un corpus contient un mélange de topics que l'on retrouve dans l'ensemble du corpus. La structure des topics est masquée, nous ne pouvons observer que les mots et documents, pas les topics eux-mêmes. Parce que la structure est cachée (ou latente), cette méthode cherche à déduire la structure des topics du corpus en fonction des mots et des documents connus.

J'ai essayé dans un premier temps d'entraîner plusieurs LDA en faisant varier le nombre de topics pour optimiser la perplexité.

J'observe un optimum pour un nombre de topics égal à 15. Toutefois, le calcul de la perplexité sur sklearn semble souffrir d'un bug qui fait qu'elle croît quand le nombre de topics augmente, alors qu'elle devrait décroître.



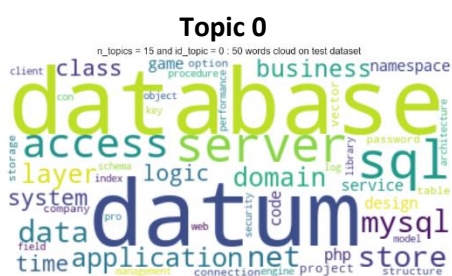
Je choisis le nombre de topics de façon à ce qu'il discrimine le mieux possible les documents. A mon sens, le meilleur compromis entre l'homogénéité de la répartition des topics dans les documents, la variété des topics et leur cohérence s'obtient avec un **nombre de topics égal à 15**.



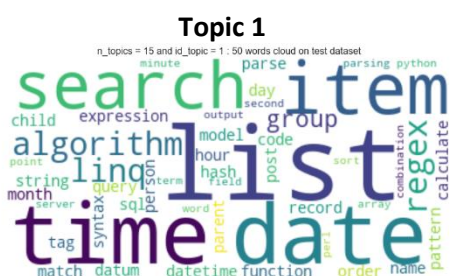
Les top 8 des mots décrivant les topics sont globalement plutôt cohérents.

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
Topic 0	database	datum	sql	server	access	mysql	data	store
Topic 1	list	time	date	item	search	algorithm	linq	regex
Topic 2	c++	c	language	session	point	compiler	code	tree
Topic 3	javascript	html	function	jquery	event	content	browser	css
Topic 4	string	property	character	path	field	format	size	system
Topic 5	windows	studio	project	process	command	script	folder	service
Topic 6	application	xml	web	.net	app	iphone	document	delphi
Topic 7	c#	php	interface	library	git	perl	code	class
Topic 8	server	test	unit	thread	memory	testing	client	connection
Topic 9	table	sql	column	row	query	datum	database	index
Topic 10	class	object	exception	code	instance	collection	constructor	reference
Topic 11	asp.net	web	site	view	http	request	config	service
Topic 12	control	image	form	button	window	ruby	wpf	variable
Topic 13	java	Project	source	eclipse	code	version	svn	subversion
Topic 14	python	Input	django	parameter	bit	procedure	function	integer

Quelques nuages de mots



Base de données



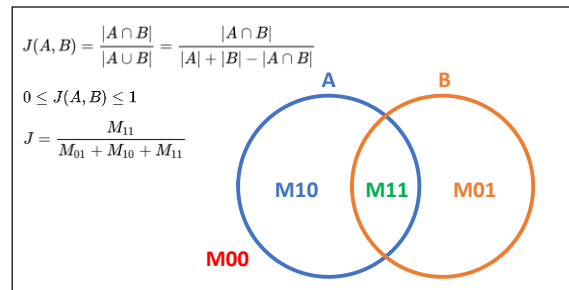
Expressions régulières et temporelles

Analyse supervisée

Indice de Jaccard

L'indice de Jaccard, défini comme étant la taille de l'intersection divisée par la taille de l'union de 2 ensembles de labels, est utilisé pour comparer un ensemble de labels prédits pour un échantillon aux labels réels correspondant.

Il s'étend aux problématiques « multilabel » en calculant une moyenne du score obtenu pour chaque label. J'utilise l'indice de Jaccard pondéré qui détermine la moyenne des métriques calculée pour chaque label, pondérée par leur distribution réelle observée.



Binary Relevance

Producing Meta Labels

dataset		binary relevance			label powerset	meta labels	
instance	labels	Y_A	Y_B	Y_C	$Y_{A,B,C}$	$Y_{A,C}$	$Y_{B,C}$
1	B	0	1	0	B	∅	B
2	B,C	0	1	1	BC	∅	BC
3	C	0	0	1	C	∅	C
4	B	0	1	0	B	∅	B
5	A,C	1	0	1	AC	AC	C
6	A,C	1	0	1	AC	AC	C
7	A,C	1	0	1	AC	AC	C
8	A,B,C	1	1	1	ABC	∅	BC
9	C	0	0	1	C	∅	C

- *binary relevance*: 9 exs, 3×2 binary classes
- *label powerset*: 9 exs, 1×5 multi-class

Dans le cas d'un problème « multilabel », il est possible d'avoir plusieurs labels pour une même instance.

Je vais utiliser l'approche Binary Relevance pour décomposer la tâche d'apprentissage multilabel en un certain nombre de tâches d'apprentissage binaires indépendantes (une par label), tout en ayant bien conscience que je préjuge de l'indépendance des labels entre eux.

Sur Sklearn, la méthode Binary Relevance est implémentée dans *OneVsRestClassifier*.

Transformation de la variable *Body* avec TermFrequency-InverseDocumentFrequency

Je transforme désormais le vocabulaire de mon corpus en une matrice de valeurs numériques en utilisant TF-IDF. Cette méthode de pondération vise à accorder une pertinence lexicale à un terme au sein d'un document. Un terme aura plus de chances d'être pertinent pour un document, si celui-ci en possède une occurrence plus élevée en son sein que les autres documents où le terme apparaît. La pertinence lexicale se mesure donc avec TF-IDF grâce à une relation entre la rareté d'un terme au sein d'un ensemble de documents et son occurrence dans un seul document.

$$w_{x,y} = \text{tf}_{x,y} \times \log \left(\frac{N}{\text{df}_x} \right)$$

TF-IDF

Term x within document y

$\text{tf}_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Etant donné la grande dimension de mon vocabulaire, je vais chercher à le réduire en optimisant les différents paramètres *min_df*, *max_df* et *max_features* de *TfidfVectorizer* à l'aide d'une recherche par

grille visant à maximiser l'indice *jaccard_weighted* qui me sert de métrique d'évaluation pour l'analyse supervisée.

Par ailleurs, je ne prends en compte que les *unigrams* et modifie le *token_pattern* par défaut pour tenir compte des mots d'un seul caractère (par exemple, le langage C).

Etant donné la grande dimension de mon vocabulaire et le nombre important de paramètres à tuner, je vais entraîner un classifieur *MultinomialNB* (multinomial Naive Bayes) peu gourmand en ressources. Le meilleur résultat est donné pour ***min_df=30, max_df=0.1*** et ***max_features=1000***, ce qui porte mon vocabulaire à **1 636 mots**, soit une matrice d'entraînement de dimension **38 821 x 1636**.

Dummy Classifier

Pour évaluer l'intérêt des modèles que j'entraîne, je vais comparer leur résultat à un classifieur naïf que j'ai entraîné à prédire en respectant la distribution des labels dans l'échantillon d'entraînement.

Jaccard weighted Train	0.024
Jaccard weighted Test	0.025

Recherche par grille du meilleur modèle

Je ne vais tester que 2 modèles étant donné la grande dimension des données et les faibles ressources computationnelles à ma disposition.

MultinomialNB

Paramètres testés	alpha : [0.0001, 0.001, 0.01, 0.1] fit_prior : [True, False]
Meilleurs paramètres	alpha : 0.001 fit_prior : False
Jaccard weighted Train	0.211
Jaccard weighted Test	0.195

Je constate une légère baisse de performance sur l'échantillon de test qui peut se traduire par un léger sur-apprentissage mais ce modèle est déjà bien plus performant que le classifieur naïf.

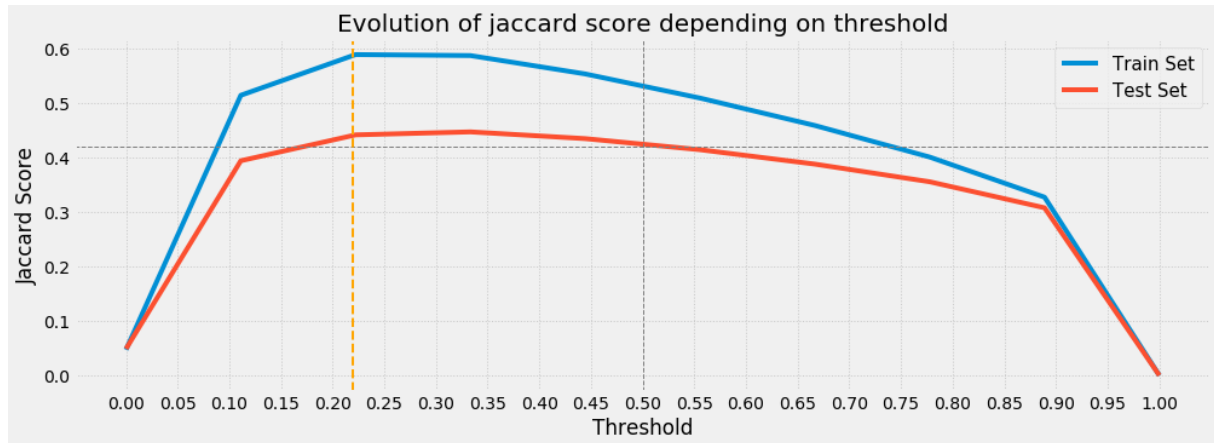
LogisticRegression

Paramètres testés	penalty : ['l1', 'l2'] C : [0.1, 1, 10, 100, 1000]
Meilleurs paramètres	penalty : l1 C : 10
Jaccard weighted Train	0.532
Jaccard weighted Test	0.424

Avec la régression logistique, la performance s'améliore mais le sur-apprentissage s'amplifie malgré la pénalisation. C'est le modèle que je vais retenir pour l'API.

Optimisation du threshold

Je teste différentes valeurs pour le seuil de probabilité visant à déterminer la classe binaire finale pour chaque label mais constate que modifier le seuil par défaut (= 0.5) n'aurait pas vraiment d'impact sur la capacité du modèle à mieux généraliser. Toutefois, je vais réduire le **seuil à 0.22**, ce qui va avoir pour effet de favoriser la capacité de mon modèle à fournir une prédiction sans en dégrader la performance.



On peut constater l'impact de la modification du seuil sur un échantillon tiré aléatoirement, le modèle est plus enclin à proposer un tag pour un seuil à 0.22 contrairement au seuil à 0.5, sans pour autant que le ou les tags proposés ne soient pas pertinents.

Y_true	Y_pred_0.5	Y_pred_0.22
(ruby, ruby-on-rails)	(performance, ruby, ruby-on-rails)	(performance, ruby, ruby-on-rails)
(c#,,)	()	(c#,,)
(version-control,,)	(algorithm,,)	(algorithm,,)
(.net, c++)	(c++,)	(c++,)
(javascript,,)	(javascript,,)	(internet-explorer, javascript)
(command-line, linux)	()	(shell,,)
(vb.net,,)	(vb.net,,)	(.net, vb.net)
(.net, user-interface, vb.net)	(.net,,)	(.net, multithreading)
(css, html, internet-explorer)	()	(java,,)
(c#, xml)	(c#, xml)	(c#, xml)

API

L'application est réalisée en utilisant Dash. C'est une application basique qui propose une liste de tags StackOverflow (jusqu'à 3 prédits par l'approche supervisée, et 5 prédits par l'approche non supervisée) relatifs à une question saisie traitant de sujets informatiques et en Anglais.

Pickles (.pkl) nécessaires :

- ignore_words (préprocessing) : liste des mots qui ne doivent pas être modifiés par le NLP
- specialtags (préprocessing) : tags contenant des caractères spéciaux (C#...)
- manual_stopwords (préprocessing) : stopwords issus de l'analyse exploratoire
- mlb (préprocessing) : multilabelbinarizer pour transformer les prédictions supervisées en libellé
- tf_unsupervised (préprocessing) : transformer TF pour l'approche non supervisée
- tfidf_supervised (préprocessing) : transformer TFIDF pour l'approche supervisée

- lda_model (recommandation) : modèle non supervisé
- lr_top100tags_3labels (recommandation) : modèle supervisé


L'ensemble des fonctions réalisées dans le cadre de ce projet ont été stockées dans un module *utils.py* :

- clean_whitespace_and_code
- apply_specialtags_transco
- clean_punctuation
- stopWordsRemove
- lemmatization
- pred_nwords_unsupervised
- recommend_tags

Avant prédiction, le texte saisi passe par toutes les étapes de preprocessing NLP évoqués en amont :

```
text = text.apply(lambda s: clean_whitespace_and_code(s))
text = text.apply(lambda s: BeautifulSoup(s).get_text())
text = text.apply(lambda s: apply_specialtags_transco(s, specialtags))
text = text.apply(lambda s: clean_punctuation(s))
text = text.apply(lambda s: stopWordsRemove(s, auto_stopwords))
text = text.apply(lambda s: lemmatization(s, ['NOUN'], ignore_words))
text = text.apply(lambda s: stopWordsRemove(s, manual_stopwords))
```

Puis il est transformé en matrice TF-IDF avant application des modèles supervisés et non supervisés.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8050'. The page title is 'Recommandation de tags StackOverFlow pour une question'. Below the title, there is a text input field containing the question: 'I want to write a simple regular expression in Python that extracts a number from HTML.'. Below the input field is a 'SUBMIT' button. Underneath the button, there are two rows of recommended tags. The first row contains 'python, regex' and the second row contains 'python, javascript, function, html, jquery'.

Pistes d'amélioration

- Pour le modèle non supervisé, retirer les mots les plus fréquents par topic pourrait permettre d'amener un peu plus de spécificité. Je pourrais gagner en spécificité en intégrant des n-grams ou en utilisant des techniques de plongements de mots pour prendre en compte le contexte.
- Il faudrait arriver à gérer l'effet de bord induit par la faible taille des documents qui n'aide pas à bien discriminer les mots qui importent dans chaque document au sein d'une matrice TF-IDF.
- Je pourrais éviter de prédire les mêmes tags entre supervisé et non supervisé et donc intégrer les tags comme stopwords pour la matrice TF-IDF.
- Au lieu d'avoir un unique modèle supervisé pour prédire l'ensemble des tags, il ne serait pas inintéressant d'isoler les tags identifiant une technologie des autres tags. Je spécialiserais alors un classifieur à prédire la technologie concernée par la question tandis qu'un autre classifieur se concentrerait plutôt à décrire la nature du problème.