|  |
| --- |
| Parcours Datascience OpenClassrooms |
| 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. |

|  |
| --- |
| Richard Durand  01/02/2020 |

Table des matières

[Récupération des données 1](#_Toc30787857)

[Analyse de la variable *Tags* 2](#_Toc30787858)

[Natural Language Processing de la variable *Body* 4](#_Toc30787859)

[Préprocessing des données avant modélisation 5](#_Toc30787860)

[Transformation de la variable Tags 5](#_Toc30787861)

[Séparation des jeux de données en jeux d’entraînement et validation 5](#_Toc30787862)

[Transformation de la variable *Body* en « bag of words » 6](#_Toc30787863)

[Analyse non supervisée 6](#_Toc30787864)

[Méthode Latent Dirichlet Allocation 6](#_Toc30787865)

[Analyse supervisée 9](#_Toc30787866)

[Indice de Jaccard 9](#_Toc30787867)

[Binary Relevance 9](#_Toc30787868)

[Transformation de la variable *Body* avec TermFrequency-InverseDocumentFrequency 9](#_Toc30787869)

[Dummy Classifier 10](#_Toc30787870)

[Recherche par grille du meilleur modèle 10](#_Toc30787871)

[MultinomialNB 10](#_Toc30787872)

[LogisticRegression 10](#_Toc30787873)

[API 11](#_Toc30787874)

[Pistes d’amélioration 12](#_Toc30787875)

# 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 :

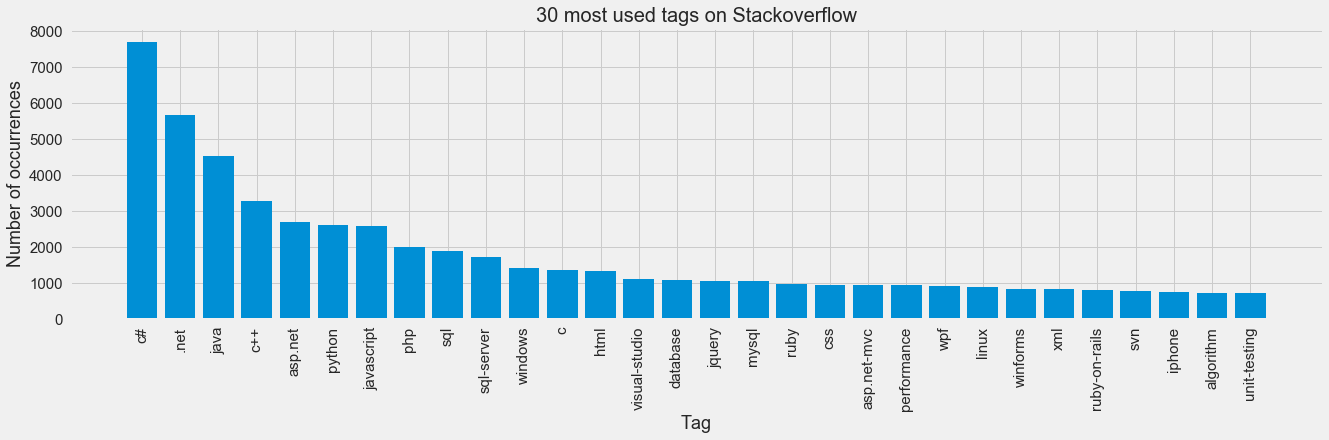
|  |  |
| --- | --- |
| Id | 121656 |
| Name | Question |
| Score | 3 |
| Body | <p>I have the following string and I would like to remove <code>&lt;bpt \*&gt;\*&lt;/bpt&gt;</code> and <code>&lt;ept \*&gt;\*&lt;/ept&gt;</code> (notice the additional tag content inside them that also needs to be removed) 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\n |
| Title | Regular expression to remove XML tags and their content |
| Tags | <c#><.net><xml><vb.net><regex> |

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*.

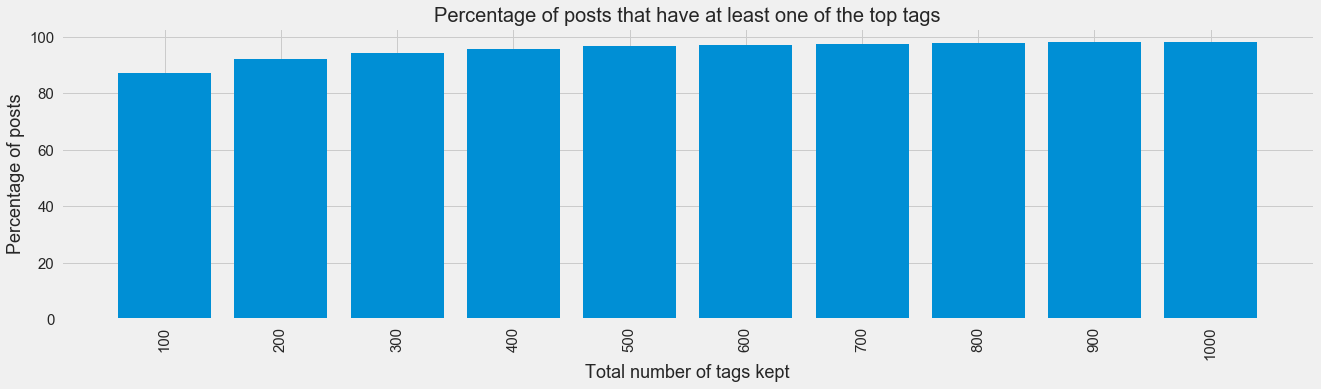
|  |  |
| --- | --- |
| C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5DD23923.tmp | La colonne Score présente une distribution asymétrique très étalée à droite avec une médiane à 4.  Pour m’assurer de ne traiter que des documents présentant un niveau de qualité suffisant, je ne conserve que ceux dont **le score est supérieur ou égal à 3, soit 55 598 documents**. |

# Analyse de la variable *Tags*

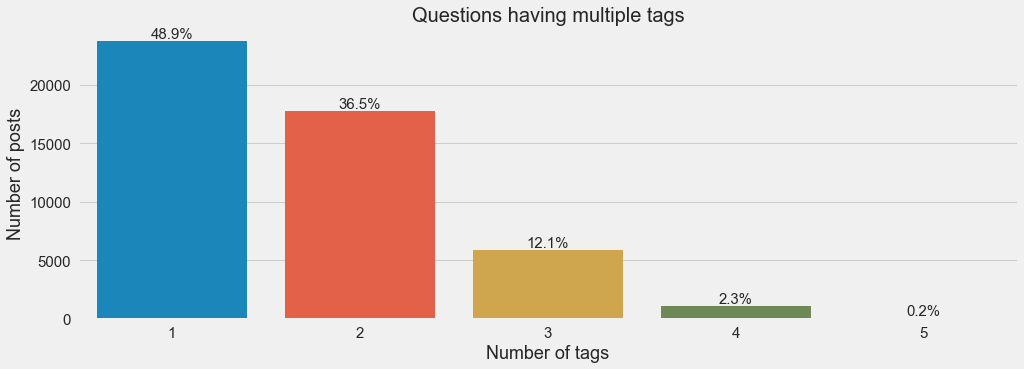
La variable *Tags* comporte **9 712 tags distincts**. Je calcule le nombre d’occurrences de chaque tag dans le dataset et produit un graphique du top 30. Le tag *<c#>* apparaît 7 703 fois et représente 9.42% des tags du dataset puis on tombe très rapidement sur des fréquences inférieures à 1% (*<unit-testing>*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



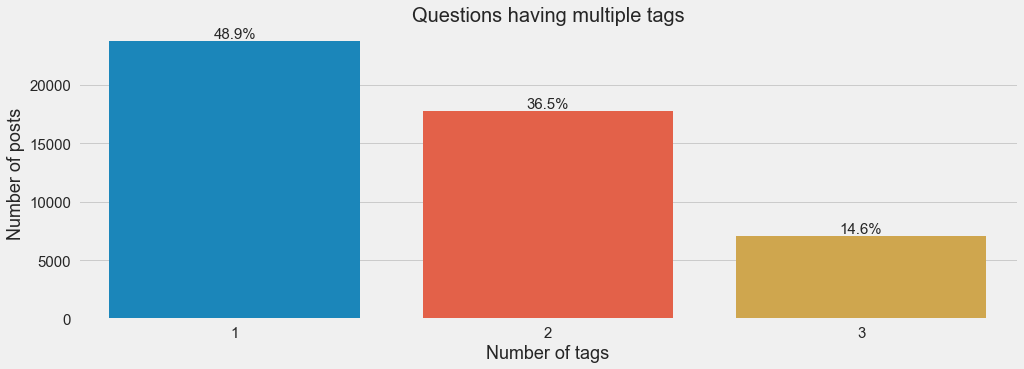
Etant donné le grand nombre de tags, je vais faire preuve de parcimonie et ne conserver qu'un nombre réduit de tags. D'une part, je m’affranchirais en partie du fléau de la dimension, cela me sera plus facile de prédire le bon tag à partir d'un nombre réduit de mots ; d'autre part, dans les presque 10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

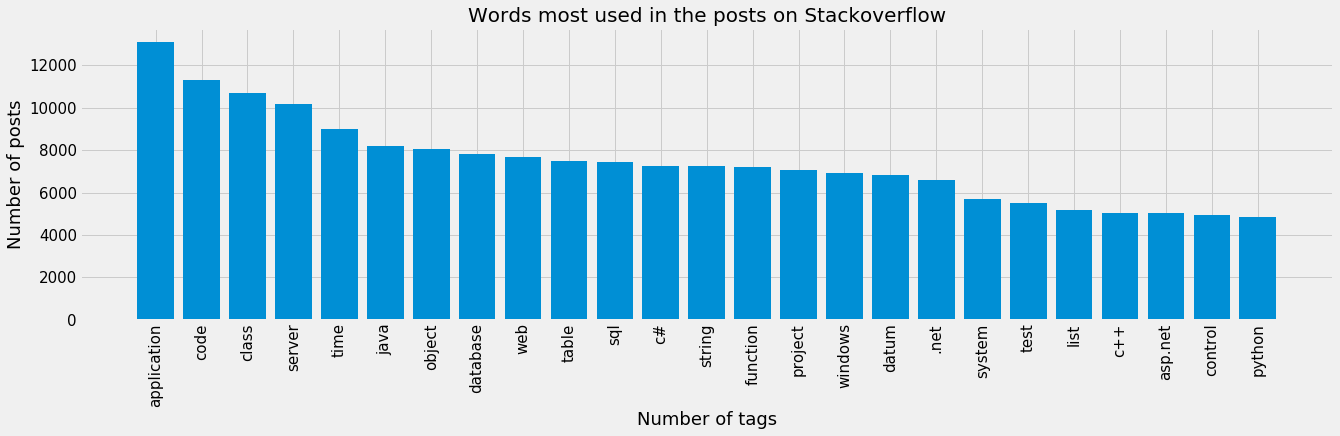
40240 string notice tag content xml parser string regex xyzspecialtags16zyx xyzspecialtags26zyx expression xml tag

* 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

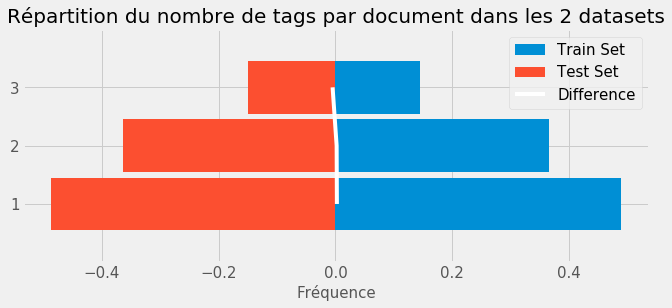
## Transformation de la variable Tags

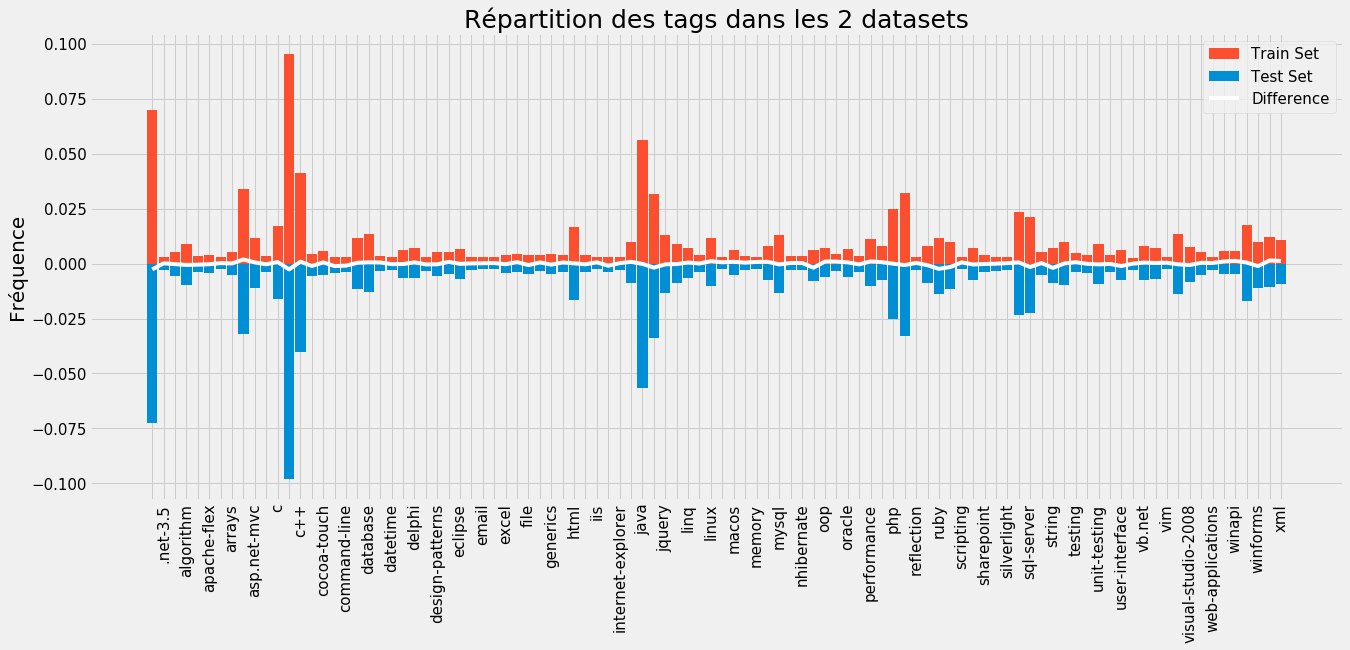
Je commence par transformer la variable *Tags* à l’aide d’un MultiLabelBinarizer pour la modélisation supervisée. J’obtiens une matrice de taille **(documents x tags) soit (48 527 x 100)** **de valeurs binaires** indiquant la présence ou non d’un ou plusieurs tags pour chaque document.



## 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).

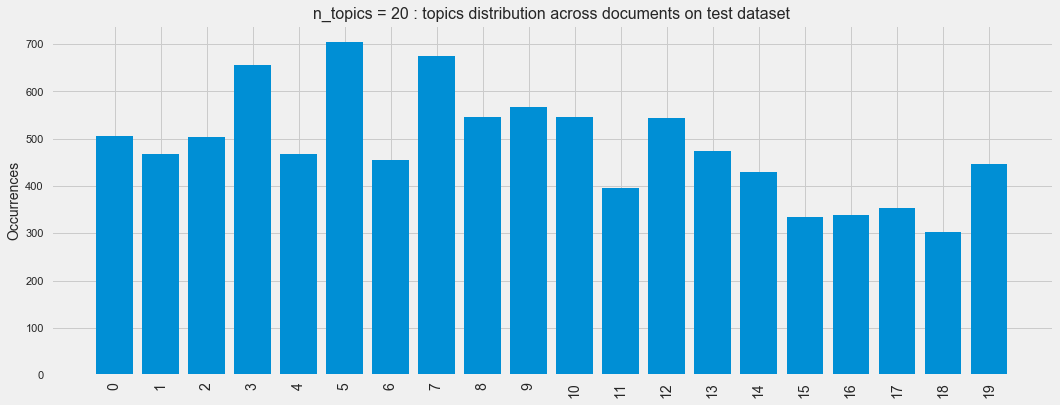
# Analyse non supervisée

## 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 donc essayé dans un premier temps d'entraîner plusieurs LDA en faisant varier le nombre de topics pour optimiser la perplexité, mais sur sklearn, il semblerait que le calcul de la perplexité souffre d'un bug qui fait qu'elle croît avec le nombre de topics, 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 20**.



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 | data | store | access | application |
| Topic 1 | List | linq | date | search | item | sql | algorithm | group |
| Topic 2 | c++ | c | language | session | point | programming | code | compiler |
| Topic 3 | Javascript | html | function | jquery | content | css | browser | ajax |
| Topic 4 | String | property | character | emacs | size | array | function | format |
| Topic 5 | Studio | project | code | version | script | command | windows | output |
| Topic 6 | Application | Xml | .net | document | delphi | performance | silverlight | flash |
| Topic 7 | c# | class | php | interface | pattern | code | function | library |
| Topic 8 | Server | windows | client | connection | machine | directory | linux | path |
| Topic 9 | Table | column | row | sql | datum | excel | query | color |
| Topic 10 | Object | class | com | thread | http | instance | collection | constructor |
| Topic 11 | asp.net | Site | view | action | controller | web | iis | website |
| Topic 12 | Control | Form | button | ruby | window | variable | item | screen |
| Topic 13 | Java | project | eclipse | source | version | code | library | software |
| Topic 14 | Python | function | sql | bit | statement | procedure | oracle | integer |
| Topic 15 | Time | memory | class | program | process | hash | performance | usage |
| Topic 16 | Test | unit | testing | git | svn | branch | code | extension |
| Topic 17 | Exception | system | windows | event | thread | process | service | application |
| Topic 18 | Image | model | email | address | django | wpf | print | field |
| Topic 19 | Web | application | service | app | perl | iphone | config | security |

|  |  |
| --- | --- |
| **Topic 0**  C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F7286E2D.tmp  Vocabulaire orienté base de données | **Topic 16**  C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5EACB387.tmp  Vocabulaire orienté tests et versioning |
| **Topic 3**  C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D6B2D417.tmp  Vocabulaire orienté développement web front | **Topic 4**  C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7667AD3D.tmp  Vocabulaire orienté type de données |
| **Topic 8**  **C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\CD24AF53.tmp**  Vocabulaire orienté architecture logicielle | **Topic 9**  **C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6B7AF7D9.tmp**  Vocabulaire orienté tableurs |
| **Topic 10**  C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\DBD1004F.tmp  Vocabulaire orienté langage objet | **Topic 15**  C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\806F9C51.tmp  Vocabulaire orienté ressources computationnelles |

Le modèle LDA appliqué à une matrice TF permet de produire 2 matrices :

* Une de dimension (d x t) contenant les probabilités des topics sachant le document
* Une de dimension (t x w) contenant les probabilités des mots sachant le topic

Je peux donc opérer un produit matriciel entre les 2 matrices pour obtenir une matrice de dimension (d x w) qui contiendra les probabilités des mots sachant le document. Il ne me restera plus qu'à choisir les N mots les plus probables pour proposer pour un document donné, les N mots les plus liés au topic latent du document.

Je réalise une fonction permettant de recommander 5 mots pour des documents passés en paramètre et observe sur quelques échantillons que si les propositions manquent de spécificité, certains termes demeurent assez adaptés.

|  |  |  |
| --- | --- | --- |
|  | **Mots proposés** | **Cleaned Body** |
| **44492** | python, c++, c, language, function | c c++ loop statement |
| **7894** | control, form, button, string, server | parentusercontrol host load parentusercontrol access property parentusercontrol childusercontrol time property parent control child control |
| **36398** | time, memory, class, program, process | class asset class class definition asset getdefinition class definition asset getdefinitionbyname |
| **17197** | java, project, c++, eclipse, source | java effect point operation java |
| **64422** | web, asp.net, application, site, object | application predispatch uri authentication store session zend request object thought authentication zend |

# Analyse supervisée

## Indice de Jaccard

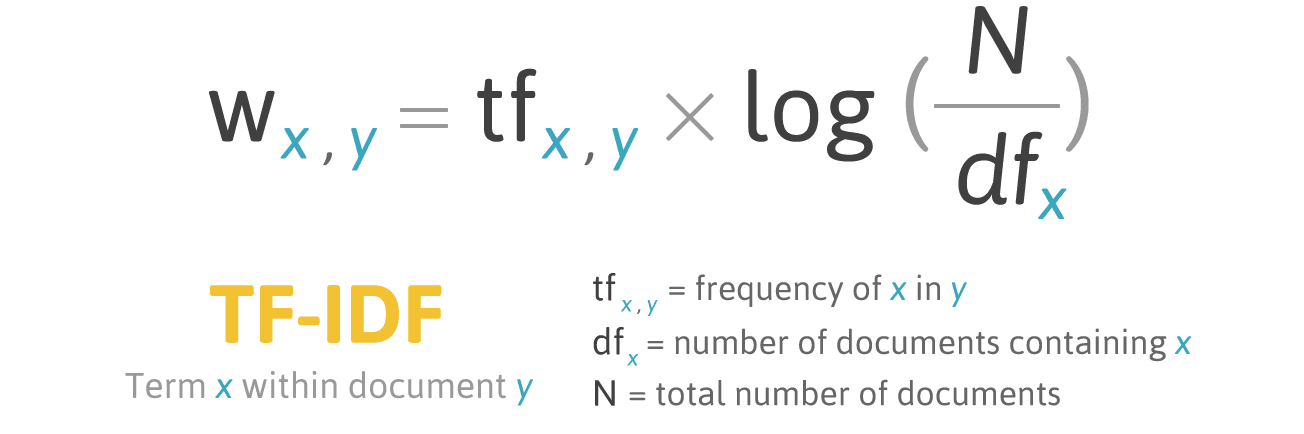
|  |  |
| --- | --- |
| L’indice de Jaccard, défini comme étant la taille de l’intersection divisée par la taille de l’union de 2 ensembles de labels, est utilisé pour comparer un ensemble de labels prédits pour un échantillon aux labels réels correspondant.  Il s’étend aux problématiques « multilabel » 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

|  |  |
| --- | --- |
| Résultat de recherche d'images pour "binary relevance multilabel" | 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.



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.

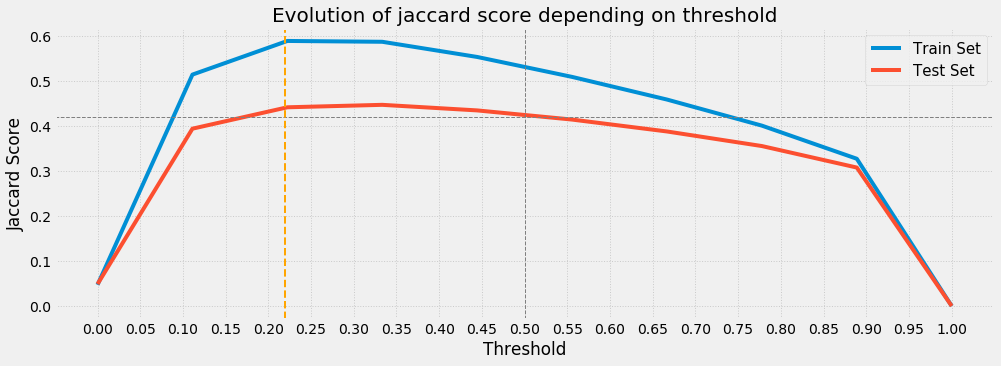
### LogisticRegression

|  |  |
| --- | --- |
| 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 préprocessing 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))

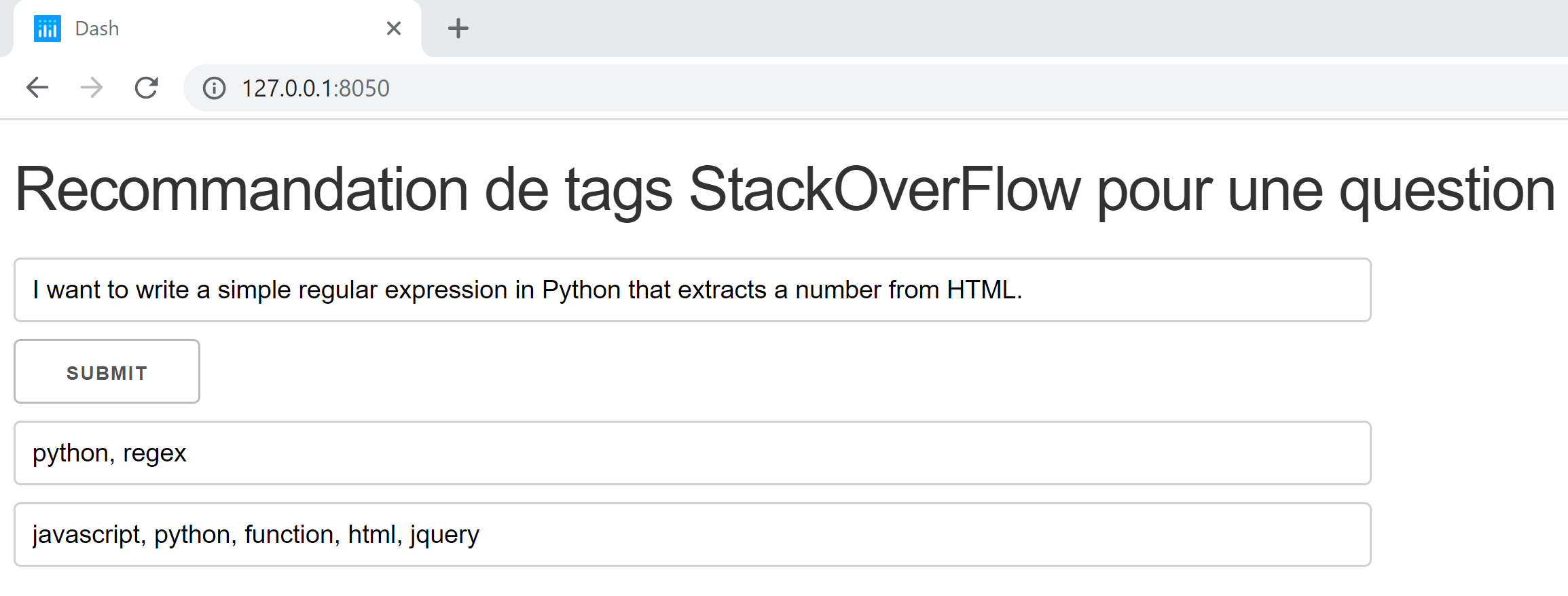
text = text.apply(lambda s: clean\_punctuation(s))

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.



# Pistes d’amélioration

* Pour le modèle non supervisé, supprimer les mots les plus fréquents par topic pourrait permettre d’amener un peu plus de spécificité. Je pourrais peut-être aussi gagner en spécificité en intégrant des n-grams ou en utilisant des techniques de plongements de mots.
* 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.