# Introduction

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 ainsi 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 pré-processer 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 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.

Enfin, 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.

# 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 91947 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 :

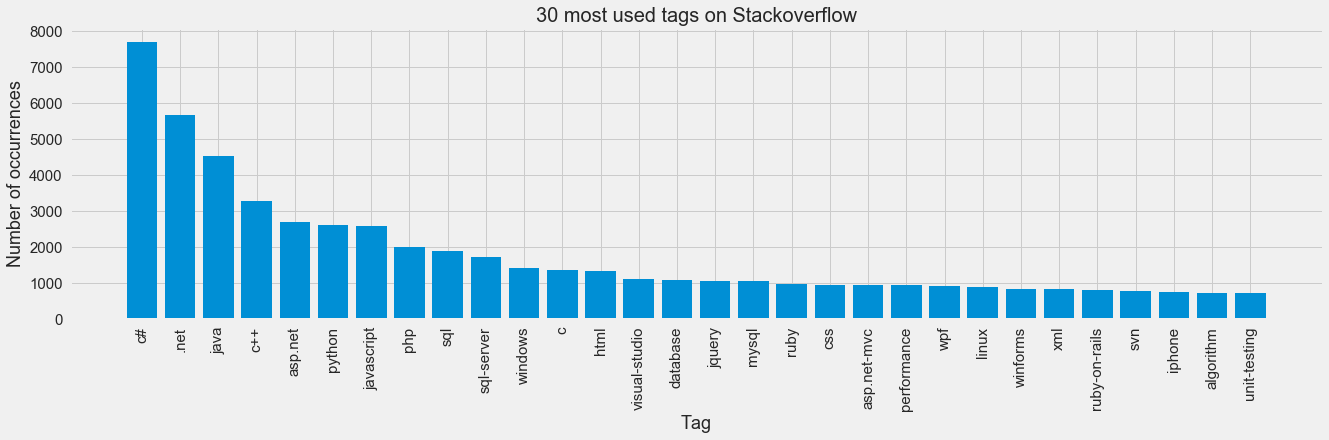
|  |  |
| --- | --- |
| Id | 121656 |
| Name | Question |
| Score | 3 |
| Body | <p>I have the following string and I would like to remove <code>&lt;bpt \*&gt;\*&lt;/bpt&gt;</code> and <code>&lt;ept \*&gt;\*&lt;/ept&gt;</code> (notice the additional tag content inside them that also needs to be removed) 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 le colonne *Title*.

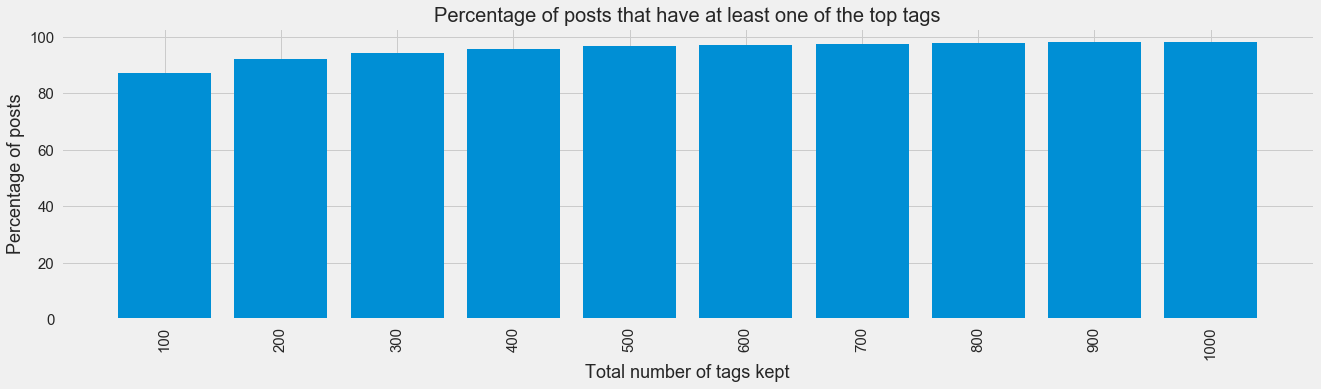
|  |  |
| --- | --- |
| C:\Users\ridur\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5DD23923.tmp | La colonne Score présente une distribution asymétrique très étalée à droite avec une médiane à 4.  Pour m’assurer de ne traiter que des documents présentant un niveau de qualité suffisant, je ne conserve que ceux dont **le score est supérieur ou égal à 3, soit 55598 documents**. |

# Analyse de la variable *Tags*

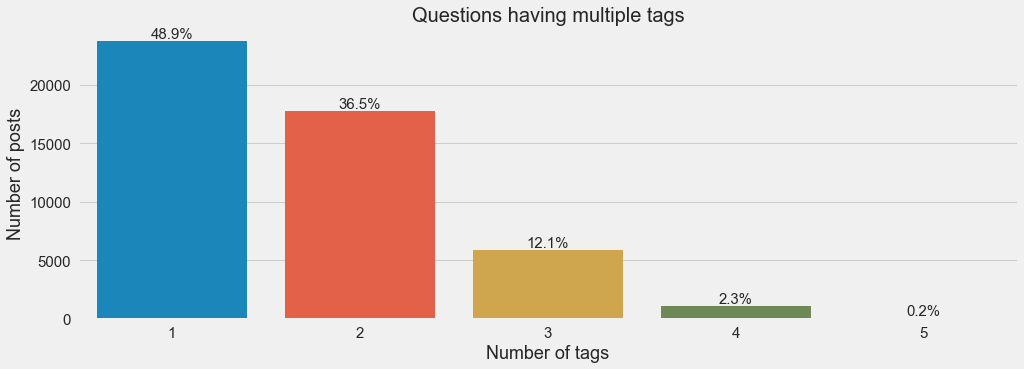
La variable *Tags* comporte **9712 tags distincts**. Je calcule le nombre d’occurrences de chaque tag dans le dataset et produit un graphique du top 30. Le tag *<c#>* apparaît 7703 fois et représente 9.42% des tags du dataset puis on tombe très rapidement sur des fréquences inférieures à 1% (*<unit-testing>* en fin de distribution sur le graphique apparaît 739 fois pour une fréquence de 0.9%)



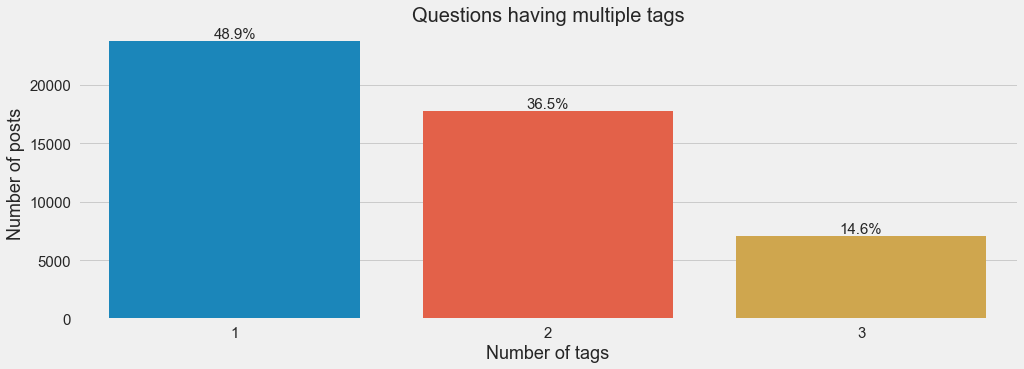
Etant donné le grand nombre de tags, je vais faire preuve de parcimonie et ne conserver qu'un nombre réduit de tags. D'une part, je m’affranchirais en partie du fléau de la dimension, cela me sera plus facile de prédire le bon tag à partir d'un nombre réduit de mots ; d'autre part, dans les presque 10000 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 **48527** **documents**.



Un document peut posséder jusqu’à 5 tags. En réalité, je constate que très peu de documents (1209) 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 stop words 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.

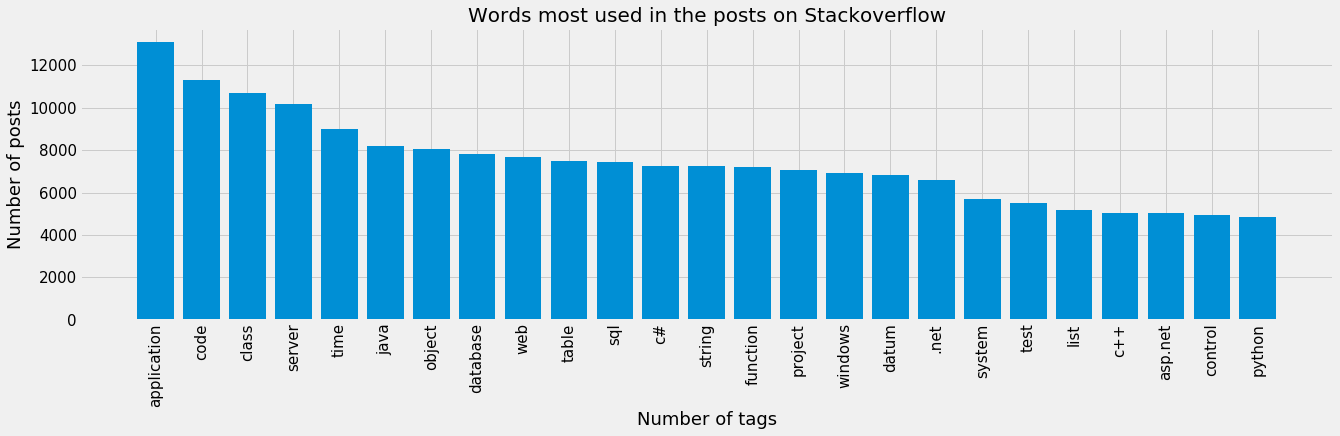
40240 string notice tag content xml parser string regex xyzspecialtags16zyx xyzspecialtags26zyx expression xml tag

* Suppression de stop words manuels. A ce state, 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

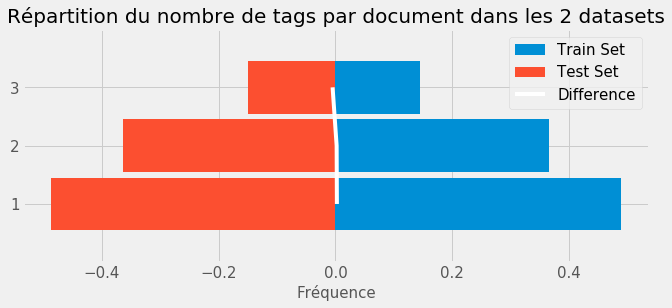
## Transformation de la variable Tags

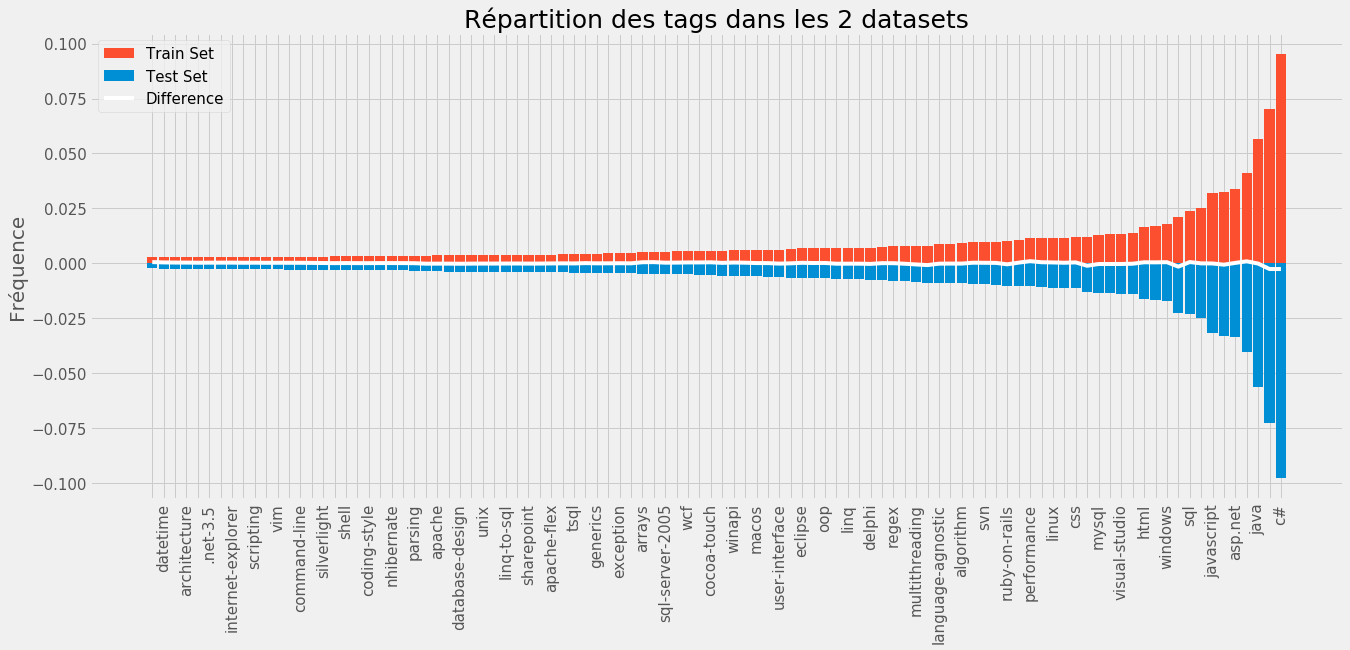
Je commence par transformer la variable *Tags* à l’aide d’un MultiLabelBinarizer pour la modélisation supervisée. J’obtiens une matrice de taille (documents x tags) soit (48 527 x 100) de valeurs binaires indiquant la présence ou non d’un ou plusieurs tags pour chaque document.



## 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.

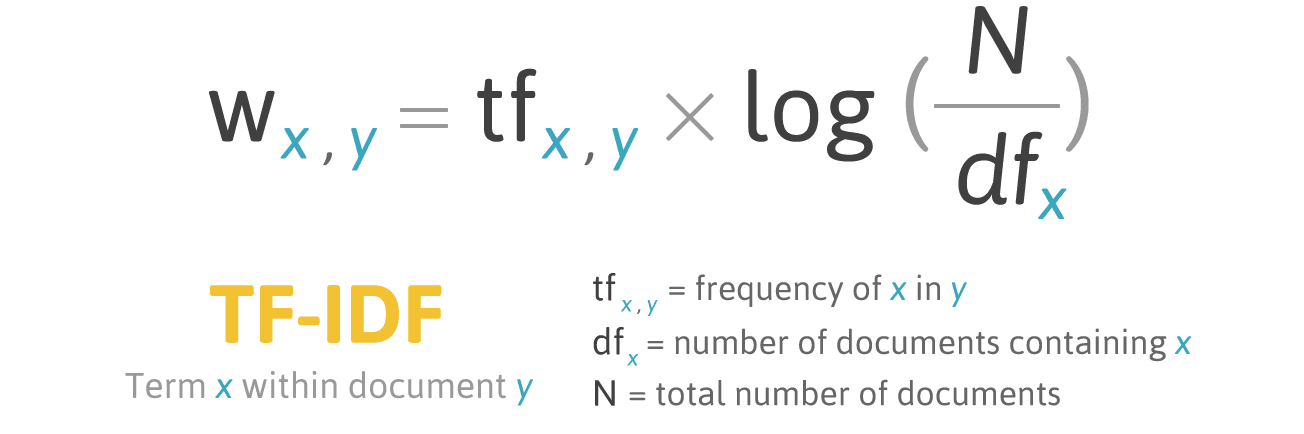




Je constate que la courbe représentant la différence entre les fréquences observées des tags dans le jeu d’entraînement et dans le jeu de validation reste stable autour de 0. Les jeux de données sont donc bien équilibrés.

## Transformation de la variable *Body* 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.



NB : une recherche par grille, intégrant la création de la matrice TF-IDF dans le pipeline de l’analyse supervisée qui va suivre, m’a permis de vérifier que modifier les valeurs par défaut des paramètres min\_df, max\_df ou max\_features de la fonction TfidfVectorizer de sklearn a un impact très limité sur les résultats de la modélisation. Je conserve donc les paramètres par défaut. J’obtiens une matrice de dimension compléter qui va me servir aux 2 analyses non supervisée et supervisée.

# Analyse non supervisée basée sur Latent Dirichlet Allocation

LDA suppose que chaque document d'un corpus contient un mélange de sujets 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.

La perplexité est une mesure statistique de la façon dont un modèle de probabilité prédit un échantillon. Appliquée à LDA, elle vise à évaluer les distributions théoriques des mots associés aux topics obtenues avec LDA, aux distributions observées des mots. Une faible perplexité indique que la distribution de probabilité est bonne pour prédire l'échantillon.